Suppose we are given a graph $G = (V, E)$ and some initial vertex labels $r_0 : V \to \mathbb{N}$. The Weisfeiler Lehman algorithm computes new vertex representations according to the formula

$$r_{i+1}(v) = \#(r_i(v), \{\{r_i(w)|w \in N(v)\}\})$$

where $\#$ is a perfect hash function, i.e. an injective function that maps its input to natural numbers.

We want to implement this Weisfeiler Lehman function using matrix vector operations to make it faster. To this end, consider the mapping $p : \mathbb{N} \to \mathbb{N}$ that maps i to the ith prime number.

Note that the prime factorization of any natural number ≥ 2 is unique. Hence, f, mapping multisets of natural numbers to numbers, defined as

$$f(\{\{r_0(w)|w \in N(v)\}\}) := \prod_{w \in N(v)} p(r_0(w))$$

is injective. This implies that f', defined as

$$f'(\{\{r_0(w)|w \in N(v)\}\}) := \sum_{w \in N(v)} \log(p(r_0(w)))$$

is injective. Now, results from Algebra tell us that $\pi = 3.14159\ldots$ can not be the result of any sum of logarithms of prime numbers. Hence

$$\#'(r_0(v), \{\{r_0(w)|w \in N(v)\}\}) := \pi \log(p(r_0(v))) + f'(\{\{r_0(w)|w \in N(v)\}\})$$

is injective.

Now note that we can write the above as a matrix vector operation

$$r_1(V) = \pi \cdot r_0(V) + A \cdot r_0(V)$$

where A is the adjacency matrix of G and $r_i(V)$ is the vector collecting the $r_i(v)$ for all $v \in V$.

Iterative Application: Weisfeiler Lehman label propagation is an iterative process that can (and should) be applied multiple times. To be able to do that, we need to find a function that transform the results $r_1(v) = \#'(r_0(v), \{\{r_0(w)|w \in N(v)\}\})$ for all $v \in V$ to the logarithms of primes. Otherwise, we cannot guarantee that $\#'(r_1(v), \{\{r_1(w)|w \in N(v)\}\})$ is injective (note that we apply $\#'$ to the representations r_1 here).

To do that, we can sort the set $\{r_1(v)|v \in V\}$ by size and assign the smallest value the smallest prime, the second smallest value the second prime, and so on. Then, we can repeat the process. To implement this efficiently in the context of vectors, we can use `numpys` `unique` function, that does just that. `pytorch` should have a very similar function.

I am not completely happy with this last part and any suggestions for alternatives are welcome.

This process was first described by Kersting et al [1].

References

[1] Kristian Kersting, Martin Mladenov, Roman Garnett, Martin Grohe (2014) Power iterated color refinement. Proceedings of the AAAI Conference on Artificial Intelligence 28(1), DOI 10.1609/aaai.v28i1.8992, URL https://ojs.aaai.org/index.php/AAAI/article/view/8992