Heuristics for Graph Neural Networks

I. Matrices

We will consider our graph defined by $G(V, E)$ with $V = \{v_i, \forall i \in [1, N]\}$ and $E = \{(e_i, e_j), \forall (i, j) \in [1, M]^2\}$. We can then define the degree as $\deg(v_i) = \sum_{j=1}^{N} A_{ij}$. From here we can define the **Adjacency matrix** as:

$$A_{ij} \triangleq \begin{cases} 1 & \text{if } (e_i, e_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Another useful definition is that of the **Degree matrix**:

$$D_{ij} \triangleq \delta_{ij} \deg(v_i) = \begin{cases} \deg(v_i) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

We can then consider the **Laplacian Matrix** defined by:

$$L_{ij} \triangleq D_{ij} - A_{ij} = \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \wedge (e_i, e_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

We say that a matrix is considered **sparse** if there are a large number of zeros making up it's entries. Whilst there is usually no strict definition for this, one common criterion is that the number of non-zero elements is roughly equal to the number of rows or columns.

II. Energy

To approach the problem at hand, we define a quantity that we call the **energy** that we wish to minimise as a measure for the quality of our solutions. Most naturally, and in accordance with [], we define our energy as:

$$E = \frac{\sum_{ij}^{N} A_{ij} c_i c_j}{N} = \frac{\sum_{i}^{N} E_i}{N} \tag{4}$$

Ideally, we hope to find a solution $\boldsymbol{c}$ that minimises the energy E. Given that there are N entries to this vector, each with 2 possible values, we have 2^N candidates. Checking the solution is equally non trivial, as we have to compare all possible obtained values. It is for this reason that the problem at hand is considered *NP-Hard*.

III. Spectral Cut Method

One way of obtaining a nice approximation for the 2-coloring problem is to use a **Spectral Cut Method**. For this, we select the eigenvector associated with the largest magnitude eigenvalue, and associate the sign of each entry to the chosen colorings of the graph. More succintly, if we define the colorings as $c_i \in \{-1, 1\}$ then we have:

$$c_i = \text{sign}(v_i^\star) \text{ where } L_{ij} v_j^\star = \lambda_{\max} v_i^\star \tag{5}$$

The reason that this method is an approximation lies in the fact that ...

Algorithm 1: Spectral cut

```
L ← A − D
v* ← max(eig(L))
c ← sign(v*)
```

IV. Simulated Annealing (SA)

Algorithm 2: My cool algorithm

```
x^0 ← uniform
for t in range n
    i ← uniform
    c_i ← not c_i
```

V. Extremal Optimization (EO)

VI. Mean-field Approximation (MFA)

An alternative approach to solving the problem considers a unianimous **effective interaction** that every vertex is considered to engage with. Mathematically that is, if we consider the energy E, we most generally have that the individual contributing factor per vertex is given by:

$$E_i = c_i \underbrace{\sum_{j}^{N} A_{ij} c_j}_{\text{Interaction}} \tag{6}$$

From here, we make an *approximation* that takes each local neighbourhood interaction term, and replaces them with:

$$\sum_{j}^{N} A_{ij} c_j \to \sum_{j}^{N} A_{ij} \langle c_j \rangle = \sum_{j}^{N} A_{ij} m \tag{7}$$

Which inherently reduces the problem given that m is now the same effective interaction across all vertices.

For the problem of interest, we might first consider making the following redefinition $\mu_i \leftarrow c_i$ to make the syntax less specific to the coloring problem. From here we are modelling the local energy contributions as:

$$E_i = \mu_i \sum_{j}^{N} A_{ij} m \tag{8}$$

For a general graph coloring problem we define our local interaction as:

$$\mu_i(k) = \prod_{j}^{N} A_{ij} \sum_{c}^{C} \mu_j(c) e^{\beta(1 - 2\delta_{ck})} \tag{9}$$

But normalisation adds an additional constraint, so for 2 coloring we have:

$$\mu_i(c_1) + \mu_i(c_2) = 1 \rightarrow \mu_i = \mu_i(c_1) = 1 - \mu_i(c_2) \quad (10)$$

And so we have:

$$\begin{aligned}\mu_i(c_1) = \prod_j^N A_{ij}\Big[&\mu_j(c_1)e^{\beta\left(1-2\delta_{c_1c_1}\right)} \\ &+\mu_j(c_2)e^{\beta\left(1-2\delta_{c_2c_1}\right)}\Big]\end{aligned} \quad (11)$$

$$\begin{aligned}\mu_i(c_1) &= \prod_j^N A_{ij}\left[\mu_j(c_1)e^{-\beta} + \mu_j(c_2)e^{\beta}\right] \\ &= \prod_j^N A_{ij}\left[\mu_j(c_1)e^{-\beta} + \big(1-\mu_j(c_2)\big)e^{\beta}\right] \quad (12) \\ &= \prod_j^N A_{ij}\left[\frac{\mu_j(c_1)}{d} + \big(1-\mu_j(c_1)\big)d\right]\end{aligned}$$

$$\mu_i = \prod_j^N A_{ij}\left[\frac{\mu_j}{d} + \big(1-\mu_j\big)d\right] \quad (13)$$

We then wish to normalise these values so:

$$\nu_i(k) = \frac{\mu_i(k)}{\sum_c^C \mu_i(c)} \quad (14)$$

$$\begin{aligned}\nu_i(c_1) &= \frac{\mu_i(c_1)}{\mu_i(c_1) + \mu_i(c_2)} \\ &= \frac{\mu_i(c_1)}{\mu_i(c_1) + 1 - \mu_i(c_1)}\end{aligned} \quad (15)$$

We define first initialise all $\mu_i \sim \mathcal{N}(0,1)$ meaning that they can take on arbitrary negative and positive values. From here we use the sigmoid function to map these values via $\sigma : \mu_i \in \mathbb{R} \rightarrow p_i \in [0,1]$. This is analagous to converting the problem into that of the probability that μ_i is positive, and therefore *spin* up. That is, we take:

$$p_i = \sigma(\mu_i) = \frac{1}{1+e^{-\mu_i}} \quad (16)$$

Now we consider local interaction that is defined by:

$$p_i = \prod_j^N A_{ij} \sum \mu_j e^{\beta(1-2\delta)} \quad (17)$$

We then define a *temperature* that we define as:

$$\tau(p_i) = \log\left(\frac{p_i}{d} + (1-p_i)d\right) - \log\left(p_i d + \frac{1-p_i}{d}\right) \quad (18)$$

VII. Belief Propogation (BP)

VIII. Graph Neural Network (GNN)

IX. Best performing heuristics

X. The BP-GNN approach