The maths behind transforming TSP to Hopfield networks

The representation of TSP using Hopfield networks lends itself to an easy adaptation for the TDTSP later on, and ultimately to get the Hamiltonian of it, as we will see. To leverage this, it makes sense to first explain and optimize some of the formalisms from the HNTSP (Hopfield-Network-TSP) paper [Hopfield, 1985].

Hopfield networks are directly inspired by the Classical Ising Spin Glass Model, it consists of n nodes on a lattice, the nodes can have values of 0 and 1 (in the Ising Model these values are spin-values of +-1 or +-1/2). The energy function of any HN can be written as:

$$E(\mathbf{v}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} T_{i,j} v_i v_j - \sum_{i=1}^{n} I_i v_i \tag{1}$$

which can be rewritten and vectorized as

$$E(\mathbf{v}) = -\frac{1}{2}(\mathbf{v}^T T \mathbf{v}) - I^T \mathbf{v} \tag{2}$$

where $\mathbf{v}$ is a vector with all the nodes' states, T is a matrix consisting of the interaction terms between the nodes and I describes the external forces acting on the nodes (network "input"). Note that (1) and (2) immediately strike a resemblance to the QUBO formulation of minimizing the energy landscape of the HN. Indeed, this is what we will leverage later on to quantify the model.

For solving the TSP, Hopfield presents the following E-function:

$$E(\mathbf{v}) = \frac{A}{2}\sum_{X}\sum_{i}\sum_{i \neq j} v_{Xi} v_{Xj} \tag{3}$$

$$+ \frac{B}{2}\sum_{i}\sum_{X}\sum_{X \neq Y} v_{Xi} v_{Yi} \tag{4}$$

$$+ \frac{C}{2}\left[\left(\sum_{X}\sum_{i} v_{Xi}\right) - n\right]^2 \tag{5}$$

$$+ \frac{D}{2}\sum_{X}\sum_{Y \neq X} d_{XY} \sum_{i} v_{Xi}(v_{Y,i+1} + v_{Y,i-1}) \tag{6}$$

where A, B, C, D are parameters that need to be adjusted for the model to favor the following conditions. (3) corresponds to inhibitory row connections, meaning there should only be a single 1 in any row. (4) corresponds to inhibitory column connections, so only one 1 in every column. These terms together force the low-energy states of the network to be a valid tour. (5) acts as an additional penalty to ensure that there are only as many 1s in $\mathbf{v}$ as there are cities in the graph. (6) is called the data-term and measures the length of the path represented by $\mathbf{v}$.

Paths are encoded in the following way:

$$v_{X,i} = 1 \text{ if city X is visited at position i, else } 0 \tag{7}$$

Note that $\mathbf{v}$ is a vector, not a matrix, but in $(3, 4, 5, 6)$ the subscripts X and i correspond to city X at position i.

The tour A-B-C-A for example would be represented as $[1, 0, 0, 0, 1, 0, 0, 0, 1]$ (The return to the first site is not denoted in the encoding).

Computing the energy in $(3, 4, 5, 6)$ is very computationally inefficient. One way to tackle this problem is to try and vectorize it and convert it to the standard form (2). The original paper does not offer much improvement, it uses the Kronecker-delta function to describe the implicitly defined T-Matrix, which does not lend itself to vectorization using modern computational tools. Instead, one can, after some mathematical magic, come up with these definitions, that are, in fact, equivalent to the original energy function (proofs of these are left as an exercise to the reader, checking these will just involve writing out the involved sums):

$$T = -A * M_1 - B * M_2 - D * M_3 - C \tag{8}$$

$$M_1 = I \otimes J \tag{9}$$

$$M_2 = \Phi^T (M_1) \Phi \tag{10}$$

$$M_3 = A \otimes \alpha \tag{11}$$

where $\otimes$ denotes the Kronecker product, I is the identity matrix, J is the corresponding hollow matrix, Φ is the square commutator matrix, A the adjacency matrix of the graph on which to solve TSP and α, finally, is a circulant matrix, with the first column being 0 everywhere expect in the second and last spot. The matrices live in the following vector spaces: $I \in \mathbb{C}^{n \times n}$, $J \in \mathbb{C}^{n \times n}$, $\Phi \in \mathbb{C}^{n^2 \times n^2}$, $A \in \mathbb{C}^{n \times n}$ and $\alpha \in \mathbb{C}^{n \times n}$. n denotes the number of nodes in the graph. Consequently, $T \in \mathbb{C}^{n^2 \times n^2}$. The linear terms can be written as a vector $K \in \mathbb{C}^{n^2}$ with $2Cn$ everywhere.

Using these, we can rewrite $(3, 4, 5, 6)$ as:

$$E(\mathbf{v}) = -\frac{1}{2} \mathbf{v}^T T \mathbf{v} - K^T \mathbf{v} + n^2 \tag{12}$$

The vectorizations therefore bring our runtime complexity from $O(n^4)$ to $O(n^2)$. The value of the energy function for legal tours is proportional to the path length (and equal if $D = 1$), since everything vanishes but the data term. The constant term at the end is merely an offset such that the energies of valied tours actually are equal to the path lengths, but it has no influence on the actual minimal states, since it is a constant offset. These valid tours correspond to points on the energy surface, which, in turn, correspond to local minima, the global minimum will then be the tour with the shortest length. Also note that the problem of minimizing (12) is now written as a QUBO! So we can map it to the Quantum Ising Model by transforming our binary variables to spin variables and replacing the v_i terms with the Pauli-Z-operator acting on site i. We thus get the Ising-Model Hamiltonian, whose Eigenstates with minimum Eigenvalue correspond to the shortest tour. Check the notebooks to try it out!

Time dependent TSP

To make this problem setting time-dependent, one has to only adjust how to compute M_3 since the encoding already holds information about the time at which any site is visited and the edge weights are, in the time-independent case, only repeated.

So let M_3 now be defined as:

$$M_3 = K(I \otimes P_1) \tag{13}$$

with I being the Identity-Matrix in $\mathbb{C}^{n\times n}$ and P_1 being the circulant matrix with zeros in the first row all over except in the last entry.

K is defined as follows:

$$K = \left[(A \otimes \Lambda_0) + \sum_{i=1}^{n} (AP_2^i \otimes \Lambda_i) \right] \left[\frac{I}{\mathbf{0}} \right] \tag{14}$$

$$\Lambda_n = (P_3^T)^n \Lambda_0 P_3^n \tag{15}$$

P_3 is the circulant matrix with a first row with zeros everywhere except the second entry being one, P_2 is the circulant matrix with zeros with the first row r^0 being defined as follows:

$$r_i^0 = \begin{cases} 1 & \text{if } i = n^2 - n \\ 0 & \text{else} \end{cases}.$$

Λ_n is just a matrix with all zeros except $(\Lambda_n)_{i,i} = 1$. This matrix is called the selector matrix.

These definitions need some explaining, which will be tackled in the following paragraphs.

To make TSP time-dependent, the original constraints for valid tour-encodings all stay the same, only the data-term, which computes the path length, needs to be adjusted. In the original version this was defined as in (6). We see no time-dependence on d_{XY} and also, that the previous and next step are both taken into account, leading to a doubling of path weights which later gets cancelled out because of the division by 2. This is done to easily represent end-effects and more compactly compute the path length. In the time-dependent case however this will not work, since the weights are not constant, so going to point Y at time $i+1$ is not the same as going to point Y at time $i-1$. Therefore, one can define the new data-term as follows:

$$S = \frac{2D}{2} \sum_{X} \sum_{Y \neq X} \sum_{i} (d^{i}_{XY}) v_{Xi} v_{Y,i+1} \tag{16}$$

We can represent this computation of S by some map $\phi : \mathbb{C}^{n^2} \to \mathbb{R}$, and we can represent this map like this:

$$\phi(\mathbf{v}) = S = \mathbf{v}^T \Phi \mathbf{v} \tag{17}$$

Let us now first consider what this map does by unfolding (16) for the case of 3 cities, a, b and c:

$S = a_1(d^1_{ab}b_2 + d^1_{ac}c_2) + a_2(d^2_{ab}b_3 + d^2_{ac}c_3) + a_3(d^3_{ab}b_1 + d^3_{ac}c_1) + b_1(d^1_{ba}a_2 + d^1_{bc}c_2) + ...$

So Φ needs to act on $\mathbf{v}$ as follows:

$$\Phi : \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ b_1 \\ b_2 \\ b_3 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \mapsto \begin{bmatrix} d^1_{ab}b_2 + d^1_{ac}c_2 \\ d^2_{ab}b_3 + d^2_{ac}c_3 \\ d^3_{ab}b_1 + d^3_{ac}c_1 \\ d^1_{ba}a_2 + d^1_{bc}c_2 \\ . \\ . \\ . \end{bmatrix} \tag{18}$$

Let us define the right side as ω. We have defined what Φ should do, the next question is how. For this, let us consider some matrix K with the following structure:

$$K = \left[\begin{array}{c|c|c} 0 & \begin{matrix} d^1_{ab} & 0 & 0 \\ 0 & d^2_{ab} & 0 \\ 0 & 0 & d^3_{ab} \end{matrix} & \begin{matrix} d^1_{ac} & 0 & 0 \\ 0 & d^2_{ac} & 0 \\ 0 & 0 & d^3_{ac} \end{matrix} \\ \hline \begin{matrix} d^1_{ba} & 0 & 0 \\ 0 & d^2_{ba} & 0 \\ 0 & 0 & d^3_{ba} \end{matrix} & 0 & \begin{matrix} d^1_{bc} & 0 & 0 \\ 0 & d^2_{bc} & 0 \\ 0 & 0 & d^3_{bc} \end{matrix} \\ \hline \begin{matrix} d^1_{ca} & 0 & 0 \\ 0 & d^2_{ca} & 0 \\ 0 & 0 & d^3_{ca} \end{matrix} & \begin{matrix} d^1_{cb} & 0 & 0 \\ 0 & d^2_{cb} & 0 \\ 0 & 0 & d^3_{cb} \end{matrix} & 0 \end{array}\right] \tag{19}$$

$

Clearly, this lends itself excellently to our computation of Φ, it is, however not the full picture. Observe the second factors in each element of ω:

$$\omega = \begin{bmatrix} d^1_{ab}b_2 + d^1_{ac}c_2 \\ d^2_{ab}b_3 + d^2_{ac}c_3 \\ d^3_{ab}b_1 + d^3_{ac}c_1 \\ d^1_{ba}a_2 + d^1_{bc}c_2 \\ . \\ . \\ . \end{bmatrix}$$

The order of occurence seems to be permuted; first b_2, then b_3, then b_1, etc. Indeed, this has an intuitive meaning, since the first distance always maps the distance between the first city (the column-block in K that we pick) and the **second** city (the row in that block). So the cities at time 2 should always be mapped to the first element in the permutation, the cities at time 3 to the second and the cities at time 1 to the last ("returning"). This means, we need another map, a permutation, that we can define as follows.

$$P^* : \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ b_1 \\ b_2 \\ b_3 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \mapsto \begin{bmatrix} a_2 \\ a_3 \\ a_1 \\ b_2 \\ b_3 \\ b_1 \\ c_2 \\ c_3 \\ c_1 \end{bmatrix} \tag{20}$$

If we just consider the a components, one can find such a transformation easily:

$$P_1 : \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \mapsto \begin{bmatrix} a_2 \\ a_3 \\ a_1 \end{bmatrix}, P_1 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \tag{21}$$

Exploiting the symmetry between a, b and c, we can extend this to act on all cities at once by defining $P^* = (I \otimes P_1)$.

Using these facts, we can finally define $\Phi = KP^* = K(I \otimes P_1)$, which directly corresponds to M_3.

But how is K computed?

Assume that the adjacency matrix of a *temporal graph* (a graph whose edge weights change over time, defined as a sequence of complete graphs G_1, G_2, G_3 with adjacency matrices being A_1, A_2, A_3) is defined as follows:

$A = \left[\begin{array}{c|c|c} A_1 & A_2 & A_3 \end{array}\right]$

Using this, we can apply a selector matrix (see (15)) on A using the Kronecker product to only get the first row elements in A, spaced out to construct K later on. We need to do this for every row of course, so after that, we select the second row and add the result onto the first one. However, here we run into a problem: To let the entries in K end up in the correct spots, we need to first permute the blocks in A such that the shift them to the left. This can be done with the previously defined P_2 and can define the whole operation formally as:

$$K' = (A \otimes \Lambda_0) + (AP_2 \otimes \Lambda_1) + (AP_2P_2 \otimes \Lambda_2) = \left[(A \otimes \Lambda_0) + \sum_{i=1}^{n} (AP_2^i \otimes \Lambda_i)\right] \tag{22}$$

By applying the selector matrices to A, we do however pick up some slack; to see why this is one can by hand do the calculations, but the end result is that we can throw out the last $n^2 - n$ entries of K', which is done by multiplying with the identity matrix in the first n spots and zeros everywhere else.

$$K = \left[(A \otimes \Lambda_0) + \sum_{i=1}^{n} (AP_2^i \otimes \Lambda_i)\right] \left[\begin{array}{c} I \\ \hline \mathbf{0} \end{array}\right] \tag{23}$$

Now, one last step to make the computation of Λ_n more efficient, we can recognize that Λ_n is just Λ_0 with columns and rows being shifted to the left n times, which is done by multiplying the previously defined P_3:

$\Lambda_1 = P_3^T \Lambda_0 P_3$
$\Lambda_2 = P_3^T P_3^T \Lambda_0 P_3 P_3$

or

$\Lambda_n = (P_3^T)^n \Lambda_0 (P_3)^n$.

These considerations all lead to the final general definition of K in (14) and therefore we can compute Φ by the composition of maps K and $(I \otimes P_1)$, so we get $\Phi = K(I \otimes P_1) = M_3$. This concludes the edits we needed to make to the data term in order to use HNs in finding minima to TDTSP. This HN can then of course be directly interpreted as a classical Ising model and, by the same procedures as for TSP, be mapped to a quantum Ising model, which is in turn used to compute the lowest Eigenvalues and -states of the Hamiltonian, that correspond to the shortest tour on the temporal graph.

Dynamic TSP

While TDTSP is a step in the right direction, it ultimately has a huge constraint on our solutions: The change in edge weights cannot depend on which edge we actually took. So, for example, if we look at the graph in the following figure, we take edge (a,b). But since it took us 10 time-units to traverse edge (a,b), it could be that the graph's underlying system is in a different state than if we took, say, edge (a,c)!

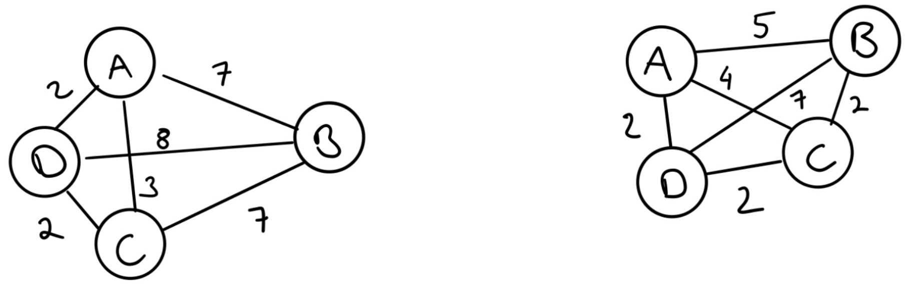

Figure 1: Time Evolution in TDTSP

A great example for this is a constellation of asteroids. If we go from asteroid A to asteroid B in 10 time-units, we would arrive at a different constellation than if we went from A to C in 5 time-units.

One way to solve this is to impose a constraint on our resting time on each asteroid: We take the edge that belongs to the shortest possible Hamiltonian cycle, but wait around until the system is in a "legal" constellation again! In our example this would mean that we take, say, (A,C) and then wait at C for 5 time-units. Then the overall time passed is 10 time-units, and we know which edge to take then. So we always wait on any asteroid until the maximum time it would have taken to get to any asteroid has passed. By doing this, TDTSP looses almost all practicality, since the actual time we need to traverse all asteroid is just the sum of all maximal edges in every graph of the sequence.

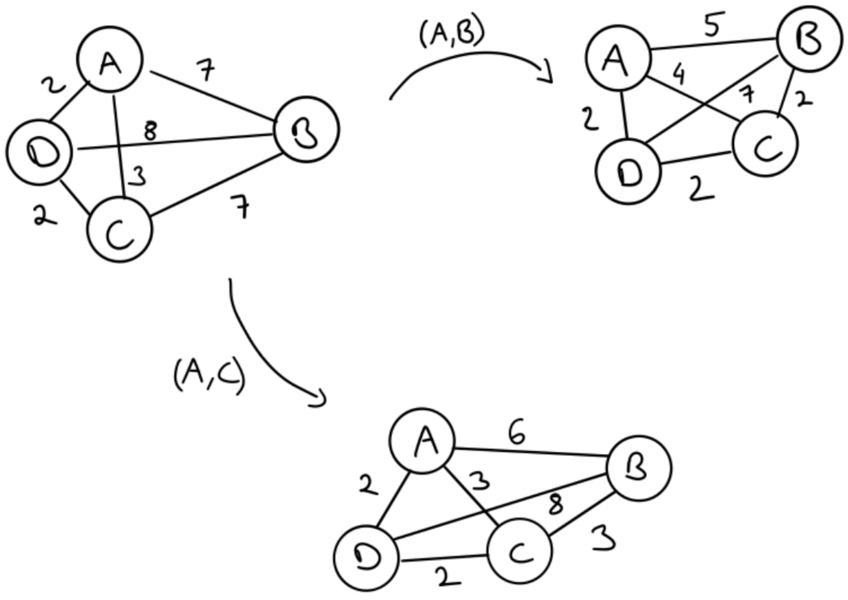

Figure 2: Time Evolution in DTSP (excerpt)

To solve this problem, we can impose a dependency of the next graph's weights on the edges we already took.

One approach is to interpret the problem of finding a tour on a given temporal Graph T consisting of a sequence of graphs $(G_1, G_2, ...)$ as finding a minimum weight n-cycle on a different graph G. A temporal graph defines a sequence of graphs with changing edge weights. To construct G, we 'unravel' every tour on T and by this construct a tree-like graph (whose leaves connect back to the root), that only consists of n-cycles. See Figure 3 for a visual. The small graph on the left corresponds to the first complete graph of T, the tree-like graph on the right is the time-unraveling of T. In this approach the first node is fixed to be A.

In our interaction matrix the following terms then have to change:

$$M_1 = I_{n_G} \otimes J_{n_T} \tag{24}$$

$$M_2 = (\Phi^{n_T, n_G})^T (I_{n_T} \otimes J_{n_G})(\Phi^{n_T, n_G}) \tag{25}$$

$$M_3 = A \otimes \alpha_{n_T} \tag{26}$$

Where n_G and n_T are number of vertices in G and T respectively and the subscripts refer to the spaces the matrices originate from, i.e. α_{n_T} is the circulant matrix of shape $n_T \times n_T$ with the second and last entry being 1 in the first row.

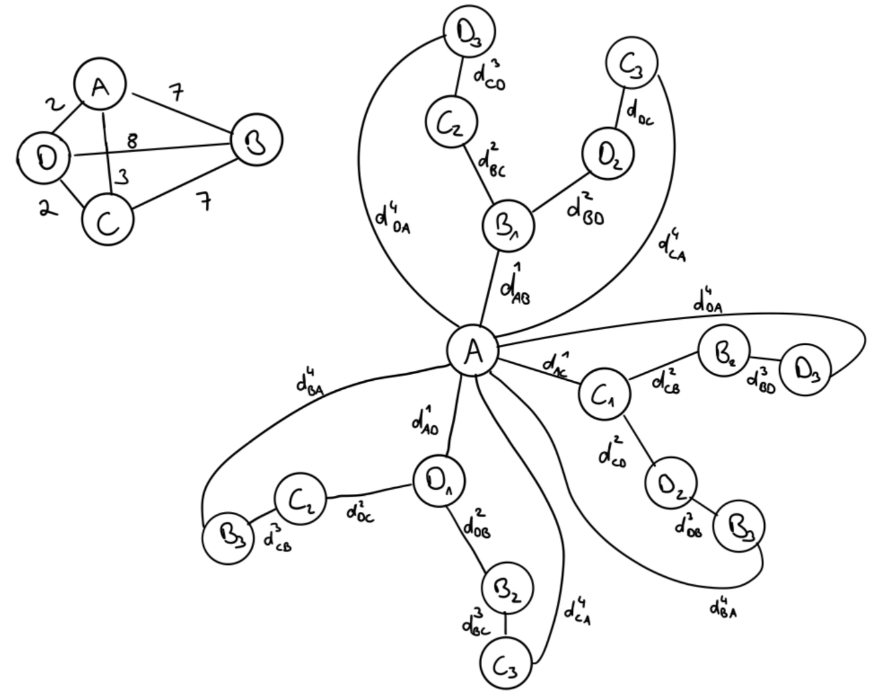

Figure 3: DTSP as n-cycle

These enforce that our (reshaped) solution of shape $(n_G * n_T) \times (n_G * n_T)$ has exactly one 1 per column and row and that the path length is minimal. A similar approach can be seen in [Bauckhage et. al., 2018], where the objective was to find the shortest path.

This solves DTSP, but the space complexity explodes, as this approach uses n_G qubits. Consider the following:

$$\begin{aligned} n_G &= 1 + (n-1) + (n-1)(n-2) + (n-1)(n-2)(n-3) + ... + 2 * (n-1)! \\ &= \sum_{i=1}^{n} \prod_{k=1}^{i} (n-k) \in \mathcal{O}(n!) \end{aligned}$$

So we need superexponentially many qubits to solve DTSP in this representation. For a tour of 10 cities for example we would need upwards of $10! = 3628800$ qubits in a quantum annealer, which is absolutely infeasible. So the goal is to find a different encoding that reduces the problem size to a more manageable qubit number, while still ensuring that we can formulate the minimization as QUBO. It seems as though this should be possible, especially since the interaction matrix is highly structured and the data term M_3 as well as the solutions are rather sparse. There does seem to be a tradeoff between denser encodings and being able to formulate the problem as a quadratic cost function. If we were to ignore this last constraint for now, we can take a look at the theoretical minimum: On a complete graph with n vertices, if we fix the first vertex, there are $(n-1)!$ tours (which correspond directly to the cycles in G). So we have an information content of $\log_2((n-1)!) \in \mathcal{O}(n \log_2 n)$ We can, for example, uniquely encode all 6 tours on a temporal graph with 4 vertices in a Bit-String of length 3. But then (this is not yet proven, but rather suspected) we cannot ensure the quadratic nature of computing E.