**3 A Series of Quantum Algorithms with Varying Complexity**

We provide three quantum algorithms with increasing algorithmic code complexity and decreasing complexity with respect to resource consumption.

**3. 1. A First Attempt**

Before presenting the algorithm, we need to explain some quantum states (M and b(\*, \*) values are as in the algorithm in section 2):

The algorithm, now, goes as follows:



α = n!; STATES = {}; N = N\_A; n = N\_B

/\* the following nested loop just before the **repeat** ... **end** command computes all subsets of n vertices in the total of N vertices of graph A so that (m\_1, m\_2, ..., m\_n) with m\_1 < m\_2 < ...< m\_n and m\_j is in {1,2, ..., N}, j in {1,2, ..., n}. \*/

**for** m\_n = N, N - 1, ..., n **do**

**for** m\_{n - 1} = N - 1, ..., n - 1 **and** m\_{n - 1} < m\_n **do**

**for** m\_1 = N - n + 1, ..., 1 **and** m\_1 < m\_2 **do**

Permutations and transpositions below apply to the tuple (m\_1, m\_2, ..., m\_n)

**repeat**

|y> = 1/sqrt(α) Σ\_{u, C} |t\_u> **χ** |C>

Measure the superposed states and let the resulting state be:



where **χ** and **Ε** denotes tensor product and inclusion in set, respectively, u **Ε** U = {u | u = k\_1 + k\_2 + ... +k\_M <= M and k\_j **Ε** {0, 1, ..., M}, 1 <= j <= M}, C **Ε** μ = {[β\_1, β\_2, ..., β\_\_{n - 1}]^T | β\_j **Ε** {0, 1}^M,1 <= i <= n - 1} and **Π** denotes composition of transpositions.

**if** γ\_{(u, C} is not in STATES **then**

Construct the permutation matrix M\_γ from the permutation γ\_{u, C}

S\_n = S\_n U **Π**\_ν(A^{β\_{1,1}}\_ν Α^{β\_{2,1}}\_ν ... A^{β\_{n - 1,ν}}\_{(n - 1)})^{τ^ν\_u}, (1 <= ν <= Μ)

STATES = STATES U {γ\_{(u, C)}}

Apply INDEX\_d\_{γ}(γ\_{u, C}, A, B, d\_γ) from the 2-Dimensional scratch tape of all possible TMs of the Quantum Machine (QM) as in section 1. It actually performs the piece of commands of the algorithm on page 20 of the paper by N. Mariella and A. Simonetto, "A Quantum Algorithm for the Sub-Graph Isomorphism Problem", (2021)

**if** d\_γ = 0 **then**

Output the permutation γ\_{u, C} at the QM's output tape

**break**

**end**

α = α - 1

**end**

**until** α = 0

**end**

**end**

**end**

Now #B = 2^{(n - 1)\*M}, #U = o(2^M) and O(M^M) (# denotes "number of"). We activate #B \* #U states totally, that is, o(2^{n \* M}) and O((2^n \* M)^M), and the length of qubit strings is o(n \* M) and O(n \* M + MlogM).

Now, the time complexity of this algorithm is obviously exponential in the worst-case scenario. We can see, however, that due to superposition and assuming the QM algorithm works on an input data augmented also by the recursive trees k\_1 + k\_2 + ...+ k\_M = ν, 1 <= ν <= M plus the matrix C, so that paths to the leaves from the roots of the recursive summation trees and, as well as, access to the elements of C, can be performed in O(1) time by the QM, and if #(d\_γ = 0) = ο(#(d\_γ <> 0)) where # denote "number of", then T(ω) = Ο(ω^c), c positive constant and ω the input length, which is defined by the N\_A and N\_B (or, samely N and n, respectively). Such a case implies that graph A is dense with regard to the isomorphic sub-graphs as defined by graph B. This is, however, not the most probable realistic instance.