Problem size scaling functions

Peter P. Rohde[1, *]
[1]*Centre for Quantum Computation and Intelligent Systems (QCIS), Faculty of Engineering & Information Technology, University of Technology Sydney, NSW 2007, Australia*
(Dated: April 22, 2018)

The computational scaling function introduced previously expresses the power of a quantum computer in terms of its classical-equivalent runtime, or equivalently FLOPs. However, this may not be the metric of interest when considering a computers algorithmic power. In many situations, of far greater interest is the size of a problem instance that can be solved in a given timespan. For example, the FLOPs associated with solving an instance of a 3-SAT problem grows exponentially with the number of clauses. When discussing the execution of this problem on a given computer, what we really want to know is how many clauses our device can cope with, rather than what the classical-equivalent runtime is.

This observation motivates us to re-parameterise the power of quantum computers in terms of the problem size of a given algorithm that can be solved. Employing the same methodology as for computational scaling functions, we define the *problem size scaling function*, which relates the size of an algorithmic problem to it's classical equivalent runtime. Then equating the computational and problem size scaling function yields,

[Problem size scaling function]

The problem size scaling function relates the size of a problem instance, in some arbitrary metric, to its classical-equivalent runtime under a time-shared model,

$$t = f_{\mathrm{size}}(s). \tag{1}$$

Equating this with the computational scaling function yields,

$$n \cdot \chi_{\mathrm{sc}}(n_{\mathrm{global}}) = f_{\mathrm{size}}(s). \tag{2}$$

Isolating the problem size yields,

$$s = f_{\mathrm{size}}^{-1}(n \cdot \chi_{\mathrm{sc}}(n_{\mathrm{global}})). \tag{3}$$

Focussing on the case where a user is in possession of a single qubit, $n = 1$, which is contributed to the time-sharing arrangement, we now consider several choice of scaling functions.

First let us consider the classical case of linear scaling functions (for both the computational and problem size scaling function),

$$\begin{aligned} f_{\mathrm{sc}}(n) &= \alpha_{\mathrm{sc}} n, \\ f_{\mathrm{size}}(s) &= \alpha_{\mathrm{size}} s \end{aligned} \tag{4}$$

Solving for the problem size simply yields,

$$s = O(1). \tag{5}$$

That is, the problem sizes of solvable instances is independent of the size of the external network with whom we are time-sharing. This is to be expected, since these scaling functions are typical of classical computers.

For polynomial scaling functions,

$$\begin{aligned} f_{\mathrm{sc}}(n) &= n^{p_{\mathrm{sc}}}, \\ f_{\mathrm{size}}(s) &= s^{p_{\mathrm{size}}}. \end{aligned} \tag{6}$$

This yields problem size,

$$s = O(\mathrm{poly}(n_{\mathrm{global}})), \tag{7}$$

demonstrating polynomial scaling in our solvable problem size against the size of the network.

For exponential scaling functions,

$$\begin{aligned} f_{\mathrm{sc}}(n) &= e^{\alpha_{\mathrm{sc}} n}, \\ f_{\mathrm{size}}(s) &= e^{\alpha_{\mathrm{size}} s}, \end{aligned} \tag{8}$$

we obtain,

$$s = O(n_{\mathrm{global}}), \tag{9}$$

demonstrating that the solvable problem size grows linearly with network size.