Bounding Graph Capacity with Quantum Mechanics and Finite Automata

Alexander Meiburg
shanqdfa@ohaithe.re

Perimeter Insitute of Theoretical Physics
Institute for Quantum Computation, University of Waterloo

Summary

The zero-error capacity of a channel (or "Shannon capacity of a graph") quantifies how much information can be transmitted with no risk of error. In contrast to the Shannon capacity of a *channel*, the zero-error capacity has not even been shown to be computable: we have no convergent upper bounds. In this work, we present a new quantity, the zero-error *unitary* capacity, and show that it can be succinctly represented as the tensor product value of a quantum game. By studying the structure of finite automata, we show that the unitary capacity is within a controllable factor of the zero-error capacity. This allows new upper bounds through the sum-of-squares hierarchy, which converges to the commuting operator value of the game. Under the conjecture that the commuting operator and tensor product value of this game are equal, this would yield an algorithm for computing the zero-error capacity.

Background: Graph Capacity

The *zero-error capacity* of a classical channel asks [Shannon, 1956] at what rate we can asymptotically transmit information with *no* chance of error. This is contrast to the standard (Shannon) capacity of the channel, which merely requires an arbitrarily small positive chance of error.

Since zero-error capacity is insensitive to the precise transition probabilities of the channel, it only depends on which input symbols can be confused for each other. This information can be described by a graph G, where two symbols $x, y \in V(G)$ are adjacent if they can be confused for each other, i.e. if they can result in the same output. For this reason, the zero-error capacity is also called the *graph capacity*, or at times, *Shannon capacity of a graph* - and written $\Theta(G)$. It has a description in terms of independent sets of the graph, and has mostly been studied as a graph theoretic problem.

One can compute convergent lower bounds on graph capacity by taking successively longer codes. But, **no convergent upper bounds are known** - as a result, **the graph capacity isn't even known to be a computable number**.

Graph Capacity $\Longrightarrow$ Regular Capacity 🤖

As a capacity, we typically think of **block codes**: codes that are decoded k symbols at a time without confusion, and then we take the limit as $k \to \infty$. We can instead think of **discrete finite automata** (DFA) that recognize the code words, with k states, and let $k \to \infty$. The codes are now **regular languages**. This gives the same capacity, since regular languages includes block codes.

Regular Capacity $\Longrightarrow$ Reversible Capacity 🔁

DFAs are given by their transition matrices, a matrix for each symbol in the language ($\Longrightarrow$ one for each vertex in the graph). These will become operators in a noncommutative polynomial optimization problem (NPO). They are 0-1 matrices, but we can't enforce that in an NPO ... but we *can* constrain unitarity. The first step is to show that *reversible* DFAs capture the full capacity.

The reversible capacity isn't necessarily equal to the regular capacity, but:

$$\Theta(G) \geq \Theta_{\mathsf{REV}}(G) \geq \Theta(G)^{\log(|G|)/(\log(|G|)+\log(\Theta(G)))} \tag{1}$$

We can **pad** the graph with useless symbols that leave $\Theta(G)$ unchanged, but squeeze Θ_{REV} arbitrarily close.

⭐ Main Result ⭐

Given a graph G, there is a noncommutative polynomial optimization problem $\mathcal{O}(G)$, with tensor-product value equal to $\Theta(G)$.

Using the NPA hierarchy, we get convergent upper bounds to the *commuting* operator value of $\mathcal{O}$; under the conjecture that this is equal to the tensor-product value, this would **prove that the graph capacity is computable**.

Reversible Capacity $\Longrightarrow$ Unitary Capacity ⚛

Now we upgrade our reversible DFA to a quantum automaton. Any classical, reversible automaton can be realized as a unitary quantum automaton, so we get at least as much capacity this way.
The hard direction is showing that quantum automata still define a (classical) code that will have an easily computable rate and easily checkable correctness (zero error rate). We get a notion of "unitary capacity" of the graph G, and prove

$$\Theta_{\mathsf{REV}}(G) \leq \Theta_U(G) \leq \Theta(G) \tag{2}$$

Accordingly, it is also squeezed arbitrarily well, and it is sufficient to compute Θ_U.

Unitary Capacity $\Longrightarrow$ $\otimes$-product NPO 📉

Encode the *correctness* and *growth rate* of a QFA-based code into an operator problem. We assume (and will enforce) that the state space is *strongly connected*.

Growth rate: like classical DFAs, this is the dominant eigenvalue of the sum of transitions - with eigenvector supported on the initial state. Strong connectedness implies the support, so this is just eigenvalue maximization.

Correctness: Run two copies of the automaton in parallel, feeding them different strings; do they ever "cheat" by accepting two confusable strings? Plays out as a *two-player quantum game*, but also one larger automaton recognizing the *perfect shuffle* of the code. "Does this automaton accept any string in this language" can be solved by *intersection* of automata and the *flood-fill algorithm*, which can again be expressed in terms of polynomial operator constraints.

We require that the players have value 1 (i.e. they're never caught cheating). We prove that a tensor-product strategy with value v implies a quantum automata for a code with capacity v. But don't know how to show this for commuting operator strategies! Showing this would *finally* give an algorithm to compute graph capacity.

Easy Example - $\Theta(G)$=2