Hamiltonian Complexity: Part 1

Sanchayan Dutta

1 November, 2022

Table of Contents

Introduction to Hamiltonian Complexity

- **Definition**: Study of Quantum Constraint Satisfaction Problems (quantum CSPs).
- **Physics**: Model a physical system as Hamiltonian. Solve Hamiltonians to make predictions.
- **Computer Science**: Hamiltonians are quantum analogues of CSPs.

Classical Constraint Satisfaction Problems

(a) k-local CSP:
- $x_1, \ldots, x_n$ variables (e.g., boolean). - $C_1, \ldots, C_m$ constraints, each C_i acts on k-variables.

Ex: 3SAT: $C_i = x_1 \vee x_2 \vee \bar{x_3}$

Goal: Find an assignment to the variables that satisfy all (or as many as possible) constraints.

Max-Cut CSPs

(b) Max-Cut CSPs

Given a graph $G = (V, E)$ partition vertices into left and right sets, to maximize # of cut edges.

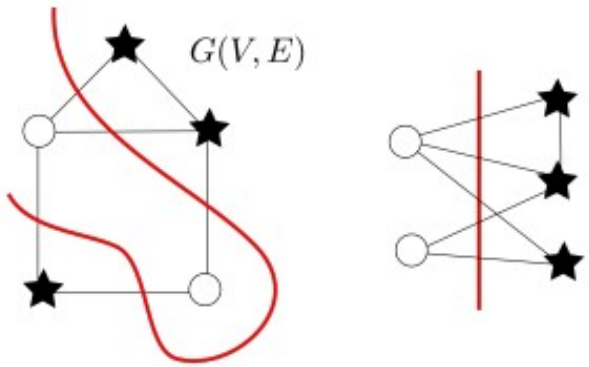


Figure 2.1: (*left*) An instance of Max-Cut on a graph $G(V, E)$ with 5 vertices and 6 edges. (*right*) The vertices are partitioned into left (circles) and right (stars) sides. The number of edges cut is 5 out of 6. There exists more than one way to achieve Max-Cut.

Max-Cut CSPs

Variables: $\{x_v\}_{v \in V}$ specifying left or right.
Constraints: $\{C_e\}_{e \in E}$
$e = (u, v)$: C_e constraint that $x_u \neq x_v$

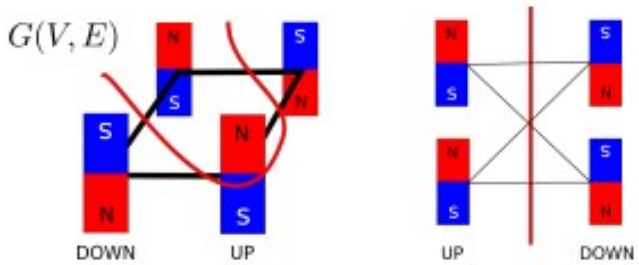


Figure 2.2: (*left*) Max-Cut instance of 2x2 grid G of magnets. All neighbouring magnets are anti-aligned. (*right*) The up and down magnets partition the vertices into left and right.

P and NP

$L \in \mathrm{P}$ if there exists a deterministic algorithm $A(x)$ s.t.
- Runs in $\mathrm{poly}(|x|)$ time.
- If $x \in L_{\text{yes}}$ then $A(x) = 1$.
- If $x \in L_{\text{no}}$ then $A(x) = 0$.

$L \in \mathrm{N}P$ if there exists a deterministic algorithm $A(x, y)$ s.t.
- Runs in $\mathrm{poly}(|x|)$ time. Here, y is the proof string.
- If $x \in L_{\text{yes}}$ then $A(x, y) = 1$.
- If $x \in L_{\text{no}}$ then $A(x, y) = 0$.

A is called the verifier algorithm.

3-SAT

3-SAT: φ instance to prove that φ is SAT, just need a satisfying assignment. If it is not SAT there is no valid satisfying assignment.

All CSPs are in NP.

The Max-Cut problem is not a decision problem by itself but we can formulate it as one. Let's define Max-Cut-0.90 as the decision problem where:
(a) L_{yes} are all graphs G that can be partitioned with $\geq 90\%$ of all edges crossing the cut.
(b) L_{no} are all graphs that cannot be partitioned in such a way.
The proof y would be a partition of a graph in G. To check, you would just have to count the number edges that are cut in the partition.

The Cook-Levin Theorem

- The famous Cook-Levin theorem (named after Steven Cook who is a professor emeritus at the University of Toronto and Leonid Levin who was in the Soviet Union at the time) says that 3SAT is NP-complete (and this is true for most CSPs).

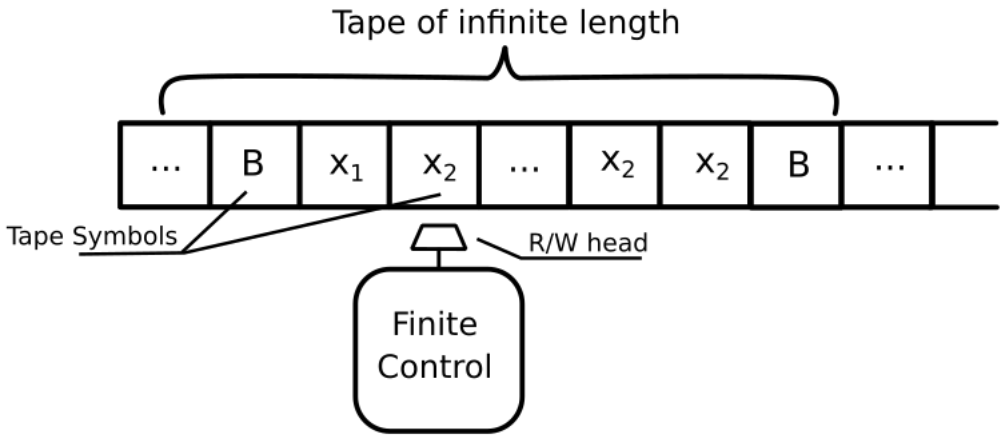
- That is, CSPs are generally complete for the class NP. In other words, not only are CSPs in NP, but they are as hard as any other problem in NP.

The Cook-Levin Theorem

More formally, given a decision problem $L \in \text{NP}$, there is an efficient way to transform instances x of L into instances φ_x of a CSP such as SAT: if $x \in L_{\text{yes}}$then φ_x is a satisfiable SAT instance, otherwise φ_x is unsatisfiable.

The Concept of Turing Machine

Every time the tape head scans a cell, the machine makes a move described in the following steps:



The finite control examines the input and changes the state according to a transition function.
The head prints any symbol of the machine alphabet (even the actual symbol) in the cell read before.
Move the tape head left or right. The head can't remain stationary, always requiring a move.

The Cook-Levin Theorem

Consider the following mental image. The algorithm A is running on a Turing machine - an idealized model of computation with a head that's moving around on a 1-dimensional memory tape, reading/writing to the tape one symbol at a time.

You have an instance x of the language L, and from the sky comes down a purported proof y. You plug x and y into the program A, and you run the program to see if it accepts (i.e. outputs 1) or rejects (output 0). Let's suppose that after $T = n^c$ time steps the Turing machine stops and accepts.

The Cook-Levin Theorem

Now suppose you'd like to convince a friend that $x \in L_{\text{yes}}$, but can't transport the Turing Machine to show them. All that is available is a description of the algorithm/Turing Machine A and the input x.

Consider taking a snapshot of the Turning Machine at each time step, where each snapshot captures the entire configuration of the machine: the contents of the tape, location of the head, state of the machine, etc. This entire sequence of snapshots $S_0, S_1, \ldots, S_T$ will be used to prove that there is some y such that $A(x, y) = 1$ and therefore $x \in L_{\text{yes}}$.

The Cook-Levin Theorem

Note that since A runs in polynomial time, all the snapshots (both individually and collectively) take up only polynomial space. Taken together, we'll call all the snapshots the computational tableau of A on input (x, y).
Now your friend can verify that that this sequence of snapshots represents the execution of A(x, y) and really does output 1.
This can be done by checking the following three conditions: **Starts OK, Evolves OK, Ends OK**.

The Cook-Levin Theorem

Starts OK: Check that in S_0, the Turing machine is properly initialized, with x written as the first input on the memory tape (there are no constraints on the second input - if some assignment can be found to the final SAT formula, that assignment will determine the proof y), and the scratch space of the algorithm (i.e. the remainder of the infinitely long tape) is set to 0.

The Cook-Levin Theorem

Evolves OK: Check that for every consecutive pair of snapshots S_t and S_{t+1}, the snapshots are consistent with each other. That is, check that if the state of the algorithm A was as described in snapshot S_t, then the snapshot S_{t+1} really would be the correct next state of the computation.

The Cook-Levin Theorem

Ends OK: Finally, check that the final snapshot S_T indicates that the final state of the algorithm is that it accepted and outputted 1. The description of the Turing Machine will have designated part of the Turing machine memory to show what its output is.

If all three of these conditions are satisfied, then there is some y such that $A(x, y) = 1$, showing that $x \in L_{\text{yes}}$.

The Cook-Levin Theorem

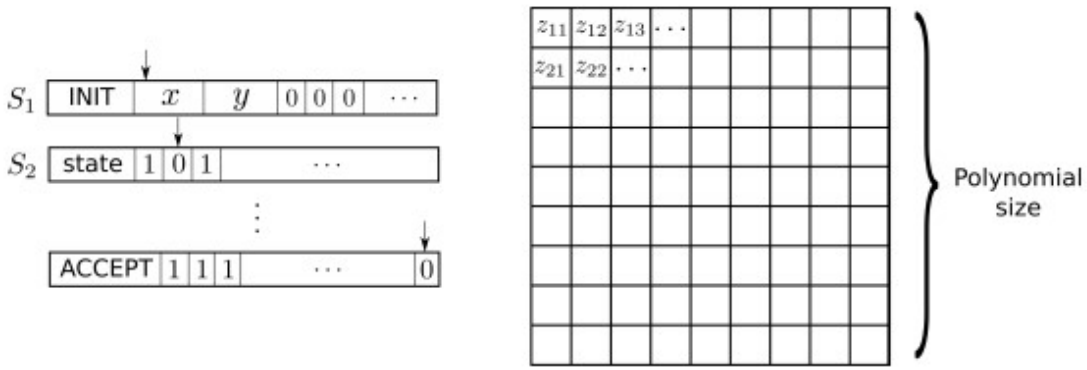


Figure 2.3: A computation tableau. On the left is a series of snapshots, capturing the state of the machine, tape and head at each step. On the right, this is transformed into a large grid of (boolean) variables from which a 3SAT instance is constructed.

The Computational Tableau

All of these conditions on the snapshots can be expressed as an instance of φ_x of 3SAT. As mentioned above, the tableau is polynomially sized: since A runs in time $T = n^c$, each snapshot is bounded in memory size by some polynomial n^c (since A runs in time $T = n^c$).

Construct a 2D grid of variables $\{z_t, i\}$ where t indicates the time step, and i indicates the i-th bit of the Turing machine state and memory. Thus there are $\mathcal{O}(Tn^c)$ total variables, and these are going to be the variables of φ_x.

The Cook-Levin Theorem

Each of the three conditions (**Starts OK, Evolves OK, Ends OK**) can be broken down into a collection of clauses on the $\{z_t, i\}$ variables. Furthermore, these clauses only have to involve 3 variables at time, so φ_x is a 3SAT formula. There's going to be a lot of clauses, but at the end of the day there will only be $\mathrm{poly}(n)$-many clauses, each with only 3 variables.

The Cook-Levin Theorem

The crucial point is that a satisfying assignment of variables $z = \{z_t, i\}$ for the 3SAT formula will essentially be a computational tableau (see figure 2.3) that proves that there is a y for which $A(x, y) = 1$, and therefore $x \in L_{\text{yes}}$. On the other hand if $x \in L_{\text{no}}$, then there is no way to satisfy all of these constraints.

The Cook-Levin Theorem

One key insight from the Cook-Levin theorem is that it relies on the fact that computation is fundamentally local: to perform any computation, you only need to change a few bits at a time in some systematic manner. This is because when constructing the constraints on $z_{t,i}$, each constraint $z_{t',i'}$ only relies on the variables immediately surrounding it on the tableau. That is what allows the Starts OK, Evolves OK, and Ends OK checks to be definable via local clauses.

The End!