Great Ideas in Computing

University of Toronto CSC196
Fall 2025

Class 16: November 3 (2025)

Announcements and Agenda

Announcements

- Our third guest presentation will be given on Monday, November 10 by Professor Niv Nayan on the subject of Data Structures.
- Assignment 3 was due this Wednesday, November 5 at 3PM. I added one thought question to the two questions previously listed. I have extended the due dat until Friday, November 7 a 3PM.
- Our fourth and final guest presenter will be Jonathan Panuelos who will discuss "computational mechanics" which relates to the simulation of physical systems. His presentation will be on Wednesday, November 19.

Today's Agenda

- A recap of Chris Maddison's presentation on machine learning. Any thoughts or questions about his presentation?
- More complexity theory. Reducing optimization andd search problems to the uderlying decision problem.

Sutton's "The Bitter Lesson"

Drew Breunig discusses the Sutton essay and says the following: Recently, "the bitter lesson" is having a moment. Coined in an essay by Rich Sutton, the bitter lesson is that, "general methods that leverage computation are ultimately the most effective, and by a large margin." Why is the lesson bitter? Sutton writes:

The bitter lesson is based on the historical observations that 1) AI researchers have often tried to build knowledge into their agents, 2) this always helps in the short term, and is personally satisfying to the researcher, but 3) in the long run it plateaus and even inhibits further progress, and 4) breakthrough progress eventually arrives by an opposing approach based on scaling computation by search and learning. The eventual success is tinged with bitterness, and often incompletely digested, because it is success over a favored, human-centric approach.

Drew Breunig: “Does the Bitter Lesson have Limits?”

The bitter lesson is an incredibly well written essay. And the argument it makes is compelling because of its simplicity.

But like any rule of thumb, it’s imperfect when it meets the real world. I embrace the idea that general methods that leverage computation will lead us to new ideas and techniques.

For many domains, once we understand these mechanisms we can apply them in focused, efficient applications. And of course – this is entirely dependent on our ability to represent our challenges as data. We can easily model chess and annotate images, but modeling workplace dynamics is much harder.

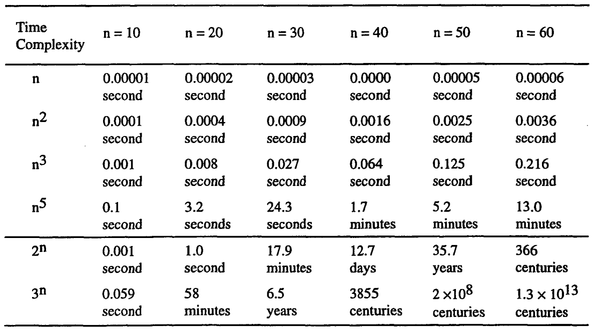
For those building agents and other AI-powered applications today, it’s good to keep in mind the bitter lesson. Consider where the model might eat your stack, but don’t be afraid to thoughtfully apply custom, human-crafted logic or embrace non-general models to get the job done. And remember: compute is a constraint. Don’t let the bitter lesson stand in the way of actually running your tool. cost is often the best way.

Table of some complexity bounding functions

A complexity theory point of view: No amount of scaling can overcome exponential complexity.

Table 2

Polynomial-Time Algorithms Take Better Advantage of Computation Time



Time Complexity	n = 10	n = 20	n = 30	n = 40	n = 50	n = 60
n	0.00001 second	0.00002 second	0.00003 second	0.0000 second	0.00005 second	0.00006 second
n^2	0.0001 second	0.0004 second	0.0009 second	0.0016 second	0.0025 second	0.0036 second
n^3	0.001 second	0.008 second	0.027 second	0.064 second	0.125 second	0.216 second
n^5	0.1 second	3.2 seconds	24.3 seconds	1.7 minutes	5.2 minutes	13.0 minutes
2^n	0.001 second	1.0 second	17.9 minutes	12.7 days	35.7 years	366 centuries
3^n	0.059 second	58 minutes	6.5 years	3855 centuries	2×10^8 centuries	1.3×10^{13} centuries

Figure: Figure taken from Garey and Johnson "Computers and intractability : a guide to the theory of NP-completeness". Time in seconds based on an estimate

How do we prove that a decision problem is *NP*-complete?

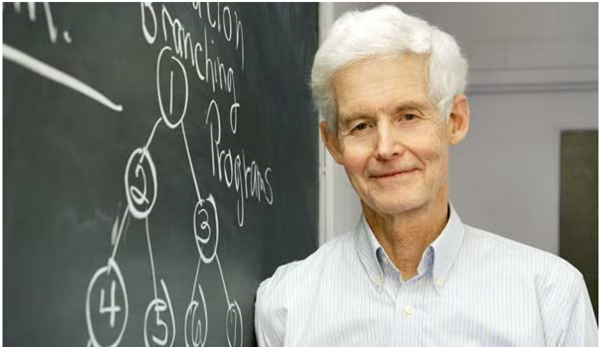
Suppose we know that some problem (for example, *SAT*) is *NP*-complete. Then if we can show *SAT* can be poly time reduced or transformed to (for example) *IS* = independent set in a graph, then *IS* must also be *NP*-complete.

Fact: Polytime reductions and polytime transformations are transitive relations. That is, for example, $A \leq_{Karp} B$ and $B \leq_{Karp} C$ implies $A \leq_{Karp} C$.

In this way, thousands of decision problems L have been created by a tree of polynomial time transformations. We will show a portion of Karp's tree of *NP* complete problems. But we have to start the tree with some *NP* problem that we prove is *NP*-complete. This "root problem" will be the *SAT* problem which was the original problem that Cook showed was *NP*-complete. He also showed that the *IndependentSet* (equivalently, the *Clique* problem) was *NP* complete by reducing *SAT* to *IndependentSet*.

Showing *SAT* is *NP* complete

Cook showed that *SAT* is *NP* complete by showing how to efficiently encode any polynomial time Turing machine computation of a verification predicate $R(x, y)$ and a "guess" for a certificate y within propositional logic. (We usually discuss this in more detail in CSC373.) Recall what we said about validity in predicate calculus (1st order logic).
Steve Cook won the Turing Award in 1982.



A simple polynomial time transformation

Let $G = (V, E)$ be a graph. An *independent set* of nodes in G is a subset $I \subseteq V$ if for all $u, v \in I$, $(u, v) \notin E$.

A *clique* in G is a subset $K \subseteq V$ if for all $u, v \in K$, $(u, v) \in E$.

We will let $\bar{S}$ for a set of edges $S \subseteq E$ denote the complement of the set. That is $(u, v) \in \bar{S}$ if and only if $(u, v) \notin S$.

NOTE: I is an independent set in G if and only if $\bar{I}$ is a clique in G.

The Independent Set and Clique problems

The Independent Set (IS) problem is the decision problem for the language $L = \{(G, k)\}$: there exists an independent set I in G such that $|I| = k\}$

The Clique problem is the decision problem for the language $L = \{(G, k)\}$: there exists a clique K in G such that $|K| = k\}$

Claim: IS $\leq_{trans}^{poly}$ Clique. Does anyone see why?

The Independent Set and Clique problems

The Independent Set (IS) problem is the decision problem for the language $L = \{(G,k)\}$: there exists an independent set I in G such that $|I| = k\}$

The Clique problem is the decision problem for the language $L = \{(G,k)\}$: there exists a clique K in G such that $|K| = k\}$

Claim: IS $\leq_{trans}^{poly}$ Clique. Does anyone see why?

For a graph $G = (V,E)$, we let $\bar{G}$ denote the complement graph $\bar{G} = (V, \bar{E})$. That is, $(u,v) \in E$ is an edge in G if and only if $(u,v) \notin \bar{G}$.

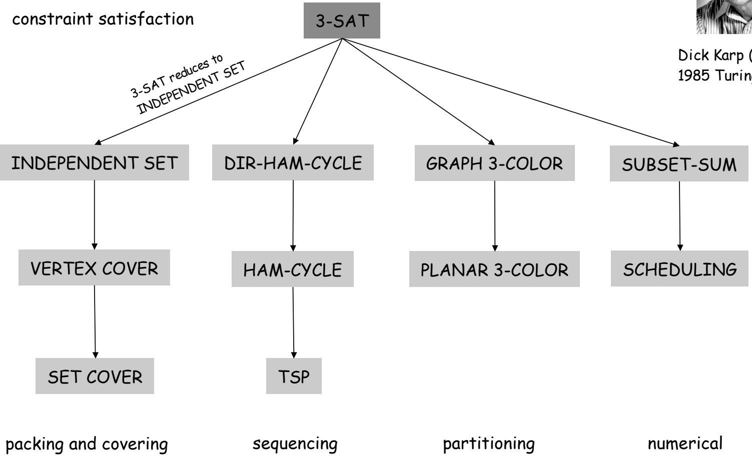
Then IS $\leq_{trans}^{poly}$ Clique by the transform h that converts (G,k) to $(\bar{G},k)$. .

Clique $\leq_{trans}^{poly}$ IS by the same transformation (i.e., taking the complement of the graph.

It is very unusual that the same transformation would work for both transformations. Some A $\leq_{trans}^{poly}$ B might be relatively easy while B $\leq_{trans}^{poly}$ A is a difficult tranformation (even when we know that such a transformation must exist). This is the case for some of the transformations in the Karp tree.

A tree of reductions/transformations

Polynomial-Time Reductions



Dick Karp (1972)
1985 Turing Award

Defining 3-SAT and exact 3SAT

Two weeks ago we defined kSat and, in particular, 3SAT. I will repeat the definition.

A propositional formula F is in conjunctive normal form (CNF) if F is presented as a conjunction $C_1 \wedge C_2 \wedge \ldots \wedge C_m$ of *clauses*. Each clause is a disjunction of *nliterals* where a literal is a propositional variable x or its complement $\bar{x}$.

Example: $F = (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_3 \vee \bar{x}_1)$.

A kSAT formula is a CNF formula where each clause has at most k literals. An exact kSAT formual is a CNF formula where each clause has exactly k literals.

The formula above is an exact 2SAT formula.

Note: It is assumed that no variable appears twice in a clause since $x \vee x \equiv x$ and $x \vee \bar{x} \equiv \mathit{TRUE}$. Here $\equiv$ is logical equivalence.
That is, $F \equiv G$ if $F|_\tau = G|_\tau$ for all truth assignments τ.

Using polynomial time reductions or transformations to derive more NP complete problems

To show that all the problems in the Karp tree are NP complete, we would need to define the transformations indicated by a $\rightarrow$ in the tree. Some of these transformations are relatively easy but some are more complicated. Each transformation is given by a polynomial time transformation h that is transforming one problem into another.

One easy (0nce you see it) transformation is thet $3SAT \leq^{p}_{trans} exact3SAT$.

Assuming that $3SAT$ or $exact3SAT$ is NP complete (or NP-hard), it follows that all the problems depicted in the tree are NP complete. The transformation only shows that all the problems in the Karp tree are NP-hard but they are also NP-complete since they are all easily seen to be in NP.)

$3SAT \leq^{p}_{trans} IS$

An example in the Karp tree of a not so obvious transformation: $3SAT \leq^{poly}_{trans}$ Independent Set. (In Cook's paper, it was stated as 3-SAT $\leq^{poly}_{T}$ Clique.)

The problems in the Karp tree are a very small sample of the thousands of NP-complete problems.
But how do we show that $3SAT$ is NP-complete? You do not have to worry about that (for this course) but here is the idea.
Suppose we have a TM $\mathcal{M}$ that we assume is executing in polynomial time (poly time in terms of the length $|w|$ of the input w to the TM). Fixing $\mathcal{M}$, the idea is to show how to encode the computation of $\mathcal{M}$ on an input w by a $3CNF$ formuala F_w. More precisely we can show:

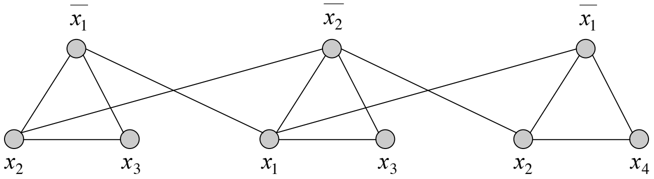
There is a polynomial time transformation h : $\mathcal{M}$ accepts w if and only if $h(w) = F_w$ is satsifiable.

3SAT reduces to Independent Set

Claim

3SAT $\leq_\tau$ Independent Set

- Given an instance F of 3SAT with k clauses, we construct an instance (G, k) of Independent Set that has an independent set of size k iff F is satisfiable.
- G contains at most 3 vertices for each clause; i.e. one for each literal.
- Connect the literals in a clause in a triangle.
- Connect a literal to each of its negations.



$$\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

Optimization problems

Each of the problems in the Karp tree has an associated optimization problem or search problem. For example, the *Vertex-Cover* problem is usually expressed as the following optimization problem:
Given a graph $G = (V, E)$, find a minimum size vertex cover for G; that is, a subset $V' \subset V$ such that for every edge $e = (u, v) \in E$, either $u \in V'$ or $v \in V'$. This is the inculusive "or" so that it is possible that both u, v are in V'.

If we can solve the optimization problem efficiently, we can immediately solve the decision problem. Does everyone understand this?

What is not as immediate, is the fact that if we can solve the *Vertex-Cover* decision problem then we can solve the *Vertex-Cover* optimization problem.

We would do this by first determining (using the decision problem) the size of the minimum vertex cover. Does everyone see how to do this?

End of Class 16

We ended disussing how optimation and search problem can be reduced to the unsderlying decision problem.

We will finish the discussion of the reduction of search and optimization problems on the Wednesday (Niv 5) class. We will then make some concluding remarks on complexitty theory.

We will then introduce complexity based cryptography.