UCT

Karp's List

Klaus Sutner

Carnegie Mellon University

Spring 2024

Cook's Theorem

In 1971, Steve Cook (a student of Hao Wang) published his seminal paper on $\mathbb{NP}$-completeness (7945 citations). Levin's work was unknown in the West at the time, thanks to the idiocy of the Cold War.

Stephen Cook

The Complexity of Theorem Proving Procedures

Proc. STACS 1971

As with the papers on Boolean functions, the perspective here is proof theory, not algorithms, and certainly not python programming.

To be clear: if Cook's paper had been confined to proof theory, no one except for few logicians would have cared one bit.

Enter Karp

Richard Karp at Berkeley[†] had a habit of reading Cook's papers–and, when he saw the 1971 paper, he realized that this was just the tip of an iceberg.

Richard Karp

Reducibility Among Combinatorial Problems

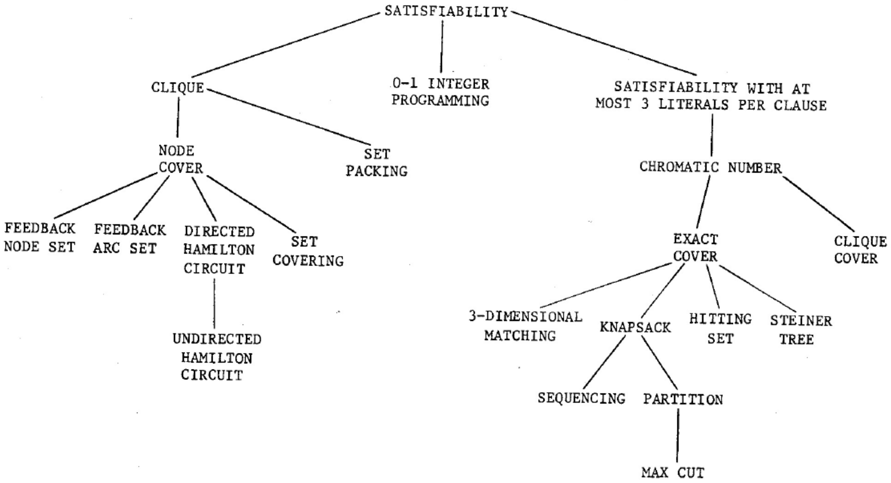
R.E. Miller, J.W. Thatcher eds., *Complexity of Computer Computations*, 1972

Karp established $\mathbb{NP}$-completeness of 21 now famous combinatorial problems that are of independent interest, beyond direct logical considerations.

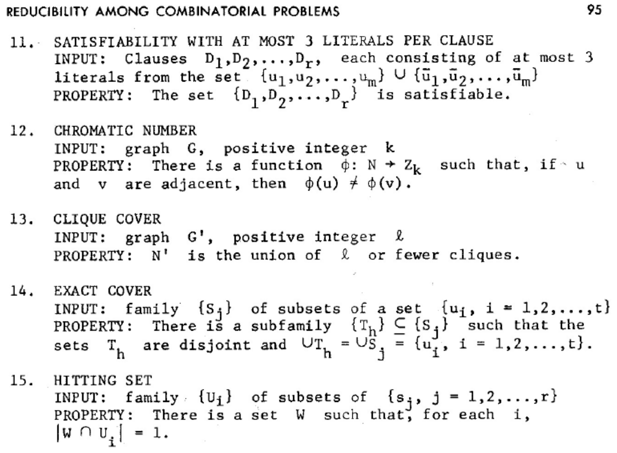
The original paper Karp 1971 is eminently readable, make sure to take a look.

[†]Berkeley denied Cook tenure, one of biggest academic blunders ever.

Roadmap


SATISFIABILITY
CLIQUE
0-1 INTEGER PROGRAMMING
SATISFIABILITY WITH AT MOST 3 LITERALS PER CLAUSE
NODE COVER
SET PACKING
CHROMATIC NUMBER
FEEDBACK NODE SET
FEEDBACK ARC SET
DIRECTED HAMILTON CIRCUIT
SET COVERING
EXACT COVER
CLIQUE COVER
UNDIRECTED HAMILTON CIRCUIT
3-DIMENSIONAL MATCHING
KNAPSACK
HITTING SET
STEINER TREE
SEQUENCING
PARTITION
MAX CUT

Problems



REDUCIBILITY AMONG COMBINATORIAL PROBLEMS 95

11. SATISFIABILITY WITH AT MOST 3 LITERALS PER CLAUSE
INPUT: Clauses $D_1, D_2, \ldots, D_r$, each consisting of at most 3 literals from the set $\{u_1, u_2, \ldots, u_m\} \cup \{\bar{u}_1, \bar{u}_2, \ldots, \bar{u}_m\}$
PROPERTY: The set $\{D_1, D_2, \ldots, D_r\}$ is satisfiable.

12. CHROMATIC NUMBER
INPUT: graph G, positive integer k
PROPERTY: There is a function $\phi: N \to Z_k$ such that, if u and v are adjacent, then $\phi(u) \neq \phi(v)$.

13. CLIQUE COVER
INPUT: graph G', positive integer ℓ
PROPERTY: N' is the union of ℓ or fewer cliques.

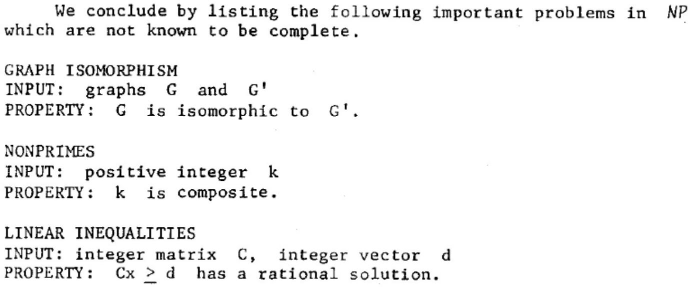
14. EXACT COVER
INPUT: family $\{S_j\}$ of subsets of a set $\{u_i, i = 1,2,\ldots,t\}$
PROPERTY: There is a subfamily $\{T_h\} \subseteq \{S_j\}$ such that the sets T_h are disjoint and $\cup T_h = \cup S_j = \{u_i, i = 1,2,\ldots,t\}$.

15. HITTING SET
INPUT: family $\{U_i\}$ of subsets of $\{s_j, j = 1,2,\ldots,r\}$
PROPERTY: There is a set W such that, for each i, $|W \cap U_i| = 1$.

Character building exercise: write a math paper using an IBM selectric.

Aside

Karp also pointed out a number of combinatorial problems that were in $\mathbb{NP}$ and obviously difficult, but not known to be complete.



We conclude by listing the following important problems in *NP* which are not known to be complete.

GRAPH ISOMORPHISM
INPUT: graphs G and G'
PROPERTY: G is isomorphic to G'.

NONPRIMES
INPUT: positive integer k
PROPERTY: k is composite.

LINEAR INEQUALITIES
INPUT: integer matrix C, integer vector d
PROPERTY: $Cx \geq d$ has a rational solution.

Update

- Primality is in $\mathbb{P}$ by Agrawal, Kayal and Saxena, 2002.

  A beautiful result using high school arithmetic, but unfortunately not practical. Probabilistic algorithms run circles around this method.

- Linear Inequalities is essentially Linear Programming, hence in $\mathbb{P}$ by a key result of L. Khachiyan in 1979.

  Similarly, Khachiyan's original method is not practical. However, there are now interier point methods that are polynomial time and are competitive with Dantzig's classical simplex algorithm at least for some instances.

- Graph Isomorphism is still a mess.

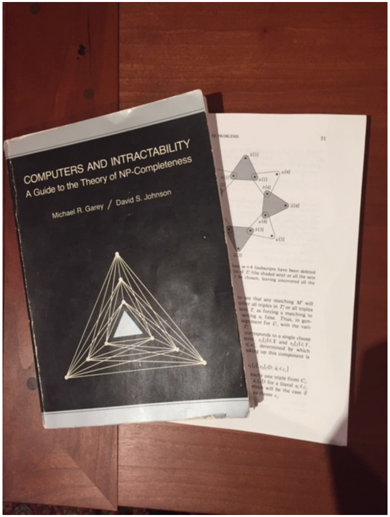
  This turns out to be the most intransigent problem. Work by Babai uses a lot of group theory in an attempt to move things towards $\mathbb{P}$, but it is not clear at this point how far one can push.

Hardness, So Far

- Satisfiability, CNF Satisfiability, 3-Satisfiability
- Vertex Cover, Independent Set, Clique
- Hamiltonian Cycle, Hamiltonian Path, Traveling Salesman Problem
- LOOP_1 Inequivalence

Here are a few more reductions that show how to enlarge the pool of $\mathbb{NP}$-complete problems, along the lines of Karp's tree.

Garey & Johnson


COMPUTERS AND INTRACTABILITY
A Guide to the Theory of NP-Completeness
Michael R. Garey / David S. Johnson
51

Sage Advice: If you are serious about complexity, you need to read this.

G&J: Six Basic Problems

- Satisfiability
- 3-Dimensional Matching
- Vertex Cover
- Clique
- Hamiltonian Cycle
- Partition

We'll do 3DM and Partition today, as well as other assorted hardness results.

Positive SAT

Define the weight of a truth assignment to be the number of variables set to true.

Problem:	**Positive SAT**
Instance:	A formula Φ in CNF, all literals positive, a bound k.
Question:	Is there a satisfying truth assignment of weight k?

Lemma

Positive SAT is $\mathbb{NP}$-complete.

Proof.

Membership is trivial, for hardness reduce from 3-SAT.

Let n be the number of Boolean variables in the instance of 3-SAT Φ'. For each literal z, introduce a new variable u_z. Set $k = n$ and define the clauses of Φ as follows:

- truth setting clauses: $\{u_x, u_{\overline{x}}\}$.
- clause clauses: for Φ' clause, say, $\{x, \overline{y}, z\}$, introduce $\{u_x, u_{\overline{y}}, u_z\}$.

A truth-assignment of weight k satisfies either u_x or $u_{\overline{x}}$, hence it exists only if Φ' is satisfiable.

The opposite direction is entirely similar.

Obviously, the Positive SAT instance Φ, k can be constructed in polynomial time. In fact, again the construction is log-space.

Our claim follows.

$\square$

Aside: Counting Problems

Our decision version of Positive SAT is clearly just a way to avoid having to talk about a natural function/counting problem:

> Give an positive CNF formula, compute the least weight of any satisfying truth-assignment.

Our hardness result for the decision version indicates that this is difficult, without having to deal with function problems. Make sure you understand how to solve the counting version given the decision version as an oracle.

Set Cover

Problem:	**Set Cover**
Instance:	A family of m subsets $S_i \subseteq [n]$, a bound k.
Question:	Is there $I \subseteq [m]$ of cardinality k such that $\bigcup_{i \in I} S_i = [n]$?

Lemma

Set Cover is $\mathbb{NP}$*-complete.*

Proof.

Membership is trivial, for hardness reduce from Vertex Cover.

Let $G = \langle V, E \rangle$ and k' be the VC instance. We may assume $V = [m]$ and $E = [n]$. Let S_i be the edges incident upon vertex i; set $k = k'$.

Cleary, $I \subseteq [m]$ such that $\bigcup_{i \in I} S_i = [n]$ corresponds to a vertex cover in G.

$\square$

Three-Dimensional Matching

Here is a 3-dimensional version of the classical matching problem on graphs.

Problem:	**Three-Dimensional Matching (3DM)**
Instance:	Three sets X, Y, Z of cardinality n, $M \subseteq X \times Y \times Z$.
Question:	Is there a matching $M' \subseteq M$ of size n?

Think of a triple (x, y, z) as a hyperedge in a hypergraph. Unfortunately, these are much harder to draw than ordinary graphs.

Matching here means that

$$\forall\, x \in X \,\exists!\, t \in M' \,(t_1 = x)$$

and likewise for the other coordinates: the chosen triples don't overlap anywhere (just like in the graph case).

Hardness

Theorem

3DM is $\mathbb{NP}$-complete.

Proof. Membership is obvious, for hardness we embed 3-SAT.

Let Ψ be an instance of 3-SAT, with n variables x_i and m clauses C_j.

A trick: to construct an instance of 3DM, we use m many incarnations of all literals x_i and $\overline{x}_i$, one for each clause.

In notation like x_{ij} we always assume $i \in [n]$, $j \in [m]$: the m-many incarnations of variable x_i.

Also, we assume that the j index wraps around: we interpret $j = m + 1$ as 1.

The Points

We need to build a collection of hyperedges $M \subseteq X \times Y \times Z$.

The Key Idea:
The middle component Y is the set of literal variants and is used to express truth assignments.
X and Z are auxiliary and will be explained in a moment.

Truth setting triples: (a_{ij}, x_{ij}, b_{ij}) and $(a_{ij}, \overline{x}_{ij}, b_{i,j+1})$

Clause triples: (s_j, x_{ij}, t_j) or $(s_j, \overline{x}_{ij}, t_j)$ if x_i or $\overline{x}_i$ is in C_j.

Garbage collection: (α, u, β) where u is a literal variant.

There are $2nm$ truth setting triples and $3m$ clause triples.

Since Y has cardinality $2nm$ we need to fill up X and Z: we simply add (meaningless) new points to satisfy the cardinality requirements of 3DM.

The sets X and Z are just the projections of these triples and both have cardinality $2mn$.

Example

For simplicity, write X for x_i, A for a_i, and B for b_i.

$$\begin{array}{ll}
(A_1, X_1, B_1) & (A_1, \overline{X}_1, B_2) \\
(A_2, X_2, B_2) & (A_2, \overline{X}_2, B_3) \\
(A_3, X_3, B_3) & (A_3, \overline{X}_3, B_4) \\
(A_4, X_4, B_4) & (A_4, \overline{X}_4, B_5) \\
(A_5, X_5, B_5) & (A_5, \overline{X}_5, B_6) \\
(A_6, X_6, B_6) & (A_6, \overline{X}_6, B_1)
\end{array}$$

An example of truth setting triples for $m = 6$. We will be forced to pick m of these triples. That means we have to pick exactly one in each row.

Critical observation: if we choose some X_j triple, the must choose all the others, we cannot select any of the $\overline{X}$ triples.

We will interpret this as choosing a truth value for $x_i = X$.

Proof

Now suppose $M' \subseteq M$ is a matching of cardinality $2nm$.

Note that M' must contain a triples of the form

$$(a_{ij}, -, -) \qquad (-, -, b_{ij}) \qquad (-, x_{ij}, -) \qquad (-, \overline{x}_{ij}, -)$$

for all i, j, But by the last observation, this means we have to pick either x_{ij}, for all j, or $\overline{x}_{ij}$, for all j.

We can interpret these choices as a truth assignment to each Boolean variable.

So far, we have only used truth setting triples. It remains to check that these assignments really work, using the other triples.

More Proof

Let's officially translate the matching M' into a truth assignment:

$$\sigma(x_i) = \begin{cases} 0 & \text{if } M' \text{ contains only } x_i \text{ triples,} \\ 1 & \text{otherwise.} \end{cases}$$

Note the flip, this is critical.

Since M' must contain one clause triple of the form $(s_j, -, -)$ for all j, it is not hard to see that σ is a satisfying truth assignment.

But the argument also works in the opposite direction: translate a given satisfying truth assignment int a corresponding matching.

Needless to say, M can be constructed in polynomial time.

□

Exact Three-Cover

Problem:	**Exact Three-Cover (X3C)**
Instance:	A family C of cardinality 3-subsets of X, $\lvert X\rvert = 3n$.
Question:	Is there an exact cover $C' \subseteq C$?

Exact cover means that each element of X appears in exactly one set in C' (so $\lvert C'\rvert = n$). In other words, C' partitions X into 3-sets.

Corollary

X3C is $\mathbb{NP}$*-complete.*

This is an easy corollary to 3DM.

Graph 3-Colorability

Problem:	**Graph 3-Colorability (G3C)**
Instance:	A ugraph G.
Question:	Is G 3-colorable?

Note that this is radically different from 2-colorability, which is easily checkable in polynomial time

Lemma

Graph 3-Colorability is $\mathbb{NP}$-complete.

Incidentally, colorability is useful for register allocation problems in systems (Chaitin, 1982). By the theorem, one has to make do with approximation algorithms.

Proof

Membership is obvious, for hardness embed 3SAT.

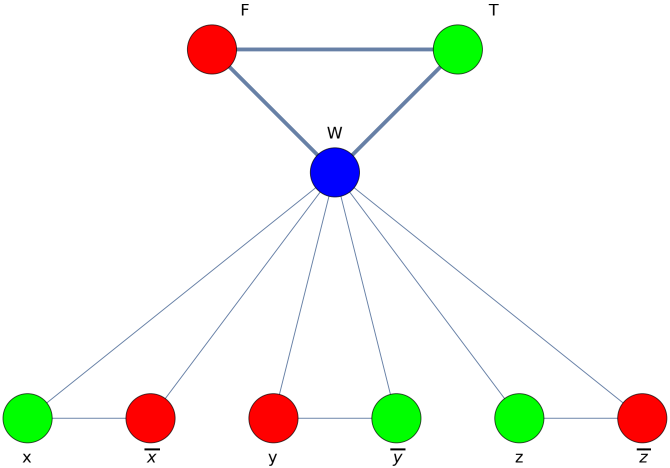
Assume Ψ is a Boolean formula with n variables $x_1, \ldots, x_n$ and m clauses C_j.

Introduce a triangle with nodes W, T, F (what, true, false).

For each variable x, there is a truth setting edge $\{x, \overline{x}\}$ and both nodes are connected to W.

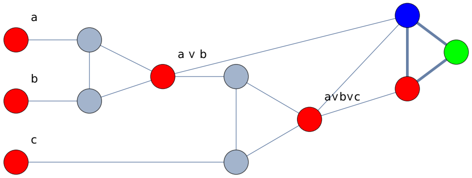
If there is a 3-coloring, we may safely assume $W \mapsto$ blue, $T \mapsto$ green, $F \mapsto$ red. Then the truth setting nodes must be either red or green.

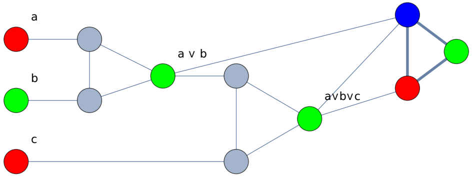
Beautiful Picture



More Proof

Truth setting nodes are also connected to “or-gates,” connections correspond to occurrence of literals in a clause $\{a, b, c\}$.





The rightmost triangle node (the output) is also connected to W and F. The first clause fails, the second is satisfied.

□

Low Degree

One "flaw" of the last construction is that it produces a graph of high degree–one might ask whether hardness holds of bounded-degree graphs.

Lemma

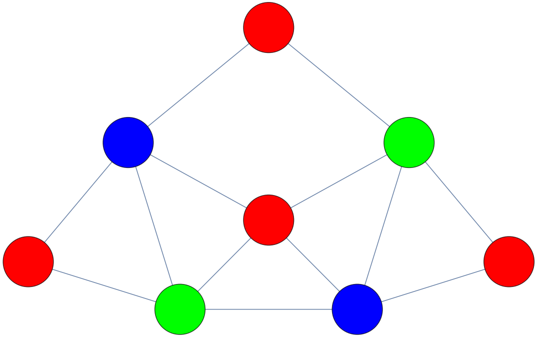
G3C is $\mathbb{NP}$-complete, even if the graph has degree at most 4.

Proof.

We replace nodes of degree $k \geq 5$ in G by little "gadgets" H_k.

H_k will have degree 4, and will have k terminal nodes to which we can connect the neighbors of the high degree vertex.

Colorability Gadgets

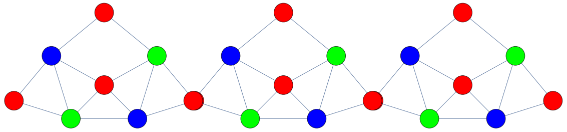


This is the gadget H_3, The terminals are the three external red nodes.

Killing High Degrees

We form H_k by chaining together $k-2$ copies of H_3, merging the left/right terminal nodes.

For example, here is H_5.



So H_k has $7(k-2)+1$ vertices, and k external terminals. By construction, H_k is 3-colorable (but not 2-colorable) and the terminals all have the same color.

We replace nodes v of G of degree $k \geq 5$ by a gadget H_k, and reroute the edges incident upon v to the terminals of H_k.

This yields a new graph H.

Clearly, 3-colorability for G is equivalent to 3-colorability of H.

$\square$

Partition into Triangles

Problem:	**Partition into Triangles**
Instance:	A ugraph $G = \langle V, E \rangle$ with $\lvert V \rvert = 3n$.
Question:	Is there a partition of V into 3-sets that all form triangles?

Lemma

Partition into Triangles is $\mathbb{NP}$-complete.

So this is a nice geometric condition: we want to partition V into blocks $\{u_i, v_i, w_i\}$, $i = 1, \ldots, n$ so that each block forms a triangle in the graph (a clique of size 3).

Proof

Membership is obvious, for hardness we embed X3C.

Recall that in X3C we have to select 3-subsets of X, $|X| = 3n$.

Write $c_i = \{x_i, y_i, z_i\} \subseteq X$ for a 3-subset, $C = c_1, \ldots, c_m$.

Start with $V = X$ and add other nodes as follows. For each $i \in [m]$, add fresh vertices a_i^j, b_i^j, c_i^j, $j \in [3]$ and add edges

$$(x_i, a_i^1), (x_i, a_i^2), (a_i^1, a_i^2), (a_i^1, a_i^3), (a_i^2, a_i^3)$$

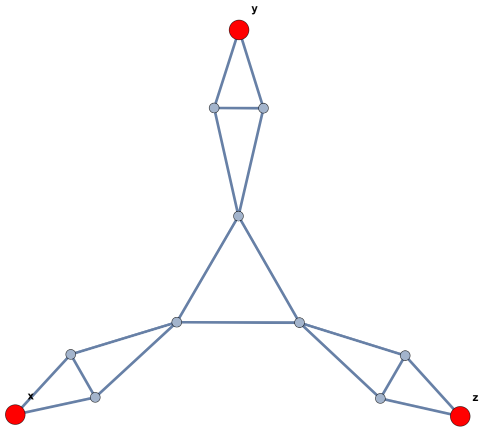
and likewise for y–b and z–c.

Lastly, add a central triangle (see pic)

$$(a_i^3, b_i^3), (a_i^3, c_i^3), (b_i^3, c_i^3)$$

This produces a 12-node gadget (3 old nodes, 9 new nodes); these can overlap only at the terminals in X.

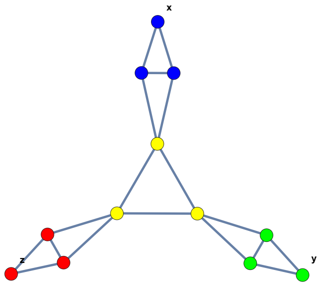
Gadget

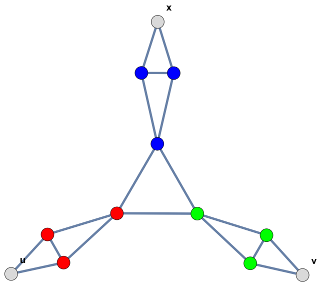


The red points are the elements of X, the rest is all scaffolding. We have one such gadget for each 3-set in C.

So?

We claim that any attempt to partition this graph into triangles is bound to use one of two methods for each gadget:





The first uses all the points in the gadget, the second does not use the terminals.

But then there is an exact 3 cover iff there is a partition into triangles.

□

Problems Involving Arithmetic

So far, our $\mathbb{NP}$-problems are purely combinatorial, arithmetic plays no role (the artificial bound introduced in some cases is not really arithmetic).

But there are other problems where numbers are an essential part of the input, and the solution may involve arithmetic operations such as addition or multiplication: for example primality testing.

Take a look at Linear Programming for a sophisticated numerical method that has lots and lots of applications.

Question: When are numbers really an essential part of the input?

Binary versus Unary

Recall that it is a sacred convention to write numbers in a instance of some decision problem in binary.

This makes perfect sense, since that it is exactly what real algorithms do: no one would dream about representing a 500-digit number in unary as input to a primality testing algorithm. Our universe is much too small for that.

Similarly all numerical algorithms rely naturally in binary representations. Fine, but that provides a handle to distinguish between problems where numbers really matter, and those where they don't: nothing much happens when we write them in unary.

Bounding Numbers

Let P be some decision problem and q a polynomial. Define the subproblem P_q to have all instances x of P such that

$$|x_{\text{unary}}| \leq q(|x|)$$

Here x_{unary} is the version of x where all numbers are written in unary, thus potentially inflating the size by an exponential amount.

For example, if we only consider TSP instances where the edge costs are 1 or 2, the inflated version x_{unary} is essentially the same as x. So here numbers do not really matter.

Definition

A problem P is strongly $\mathbb{NP}$-hard if P_q is $\mathbb{NP}$-hard for some polynomial q.

Strong $\mathbb{NP}$-Completeness

Thus, in a strongly $\mathbb{NP}$-hard problem the size of the numbers does not matter much. Even if we write the numbers in unary, the problem is intractable.

Definition

P is solvable in pseudo-polynomial time if it is solvable in time polynomial in $|x_{\text{unary}}|$, rather than its actual size $|x|$.

Claim ($\mathbb{P} \neq \mathbb{NP}$)

No strongly $\mathbb{NP}$-complete problem admits a pseudo-polynomial solution.

Example

All non-arithmetic $\mathbb{NP}$-hard problems are strongly so.
TSP is strongly $\mathbb{NP}$-hard.
Partition (next slide) is solvable in pseudo-polynomial time.

Partition

Here is a typical arithmetic problem.

Problem:	**Partition**
Instance:	A list $a_1, a_2, \ldots, a_n$ of positive integers.
Question:	Is there a subset $I \subseteq [n]$ such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

For simplicity, write $a(I)$ for $\sum_{i \in I} a_i$ whenever $I \subseteq [n]$. So we are looking for I such that $a(I) = a([n] - I)$.

Theorem ($\mathbb{P} \neq \mathbb{NP}$)

Partition is $\mathbb{NP}$-complete, but not strongly so.

Proof

Main Idea: We will use the bits of the a_i as a data structure. Since a_i will be huge, there are lots of bits to do this. Alas, we only have addition to check properties of the data structure, so this will be a bit tricky. Here goes.

Membership is obvious, for hardness we embed 3DM.

Consider an instance $M \subseteq X \times Y \times Z$ of 3DM. Let $m = |M|$ and $k = |X| = |Y| = |Z|$. We may safely assume that $[k] = X = Y = Z$, and that there are three functions $f, g, h : [m] \to [k]$ that enumerate M:

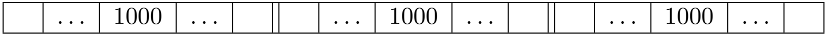
$$M = \{ (f(i), g(i), h(i)) \mid i \in [m] \}$$

The Numbers

The index set for Partition is $[n] = [m] \cup \{\alpha, \beta\}$. To describe the integers a_i, let $p = \lceil \lg m \rceil + 1$ and set for $i \in [k]$;

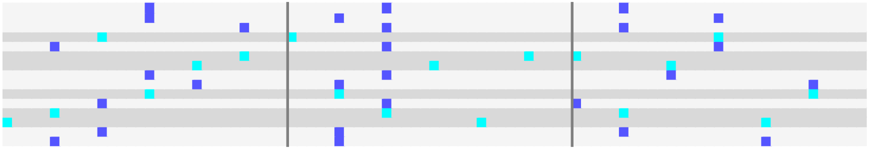
$$a_i = 2^{p(2k+f(i))} + 2^{p(k+g(i))} + 2^{p\,h(i)}$$

Here is a picture of such a number. It is a giant bitvector for sets X, Y and Z, but the bits are padded to length p.



Thus, a_i has exactly three 1 bits, and each appears in one of k many positions. The possible positions are spaced out to be a multiple of p.

Example



A 3DM Yes-instance:

$$((3,4,5),(3,5,3),(1,4,5),(4,6,3),(5,4,3),(1,1,6),(2,3,4),(3,4,4),\\ (2,5,1),(3,5,1),(4,4,6),(5,4,5),(6,2,2),(4,5,5),(5,5,2))$$

Here $m = 15$, $k = 6$.

Each of the critical columns contains exactly one cyan block. The corresponding rows represent the matching.

More Proof

Set $S = \sum_{i=1}^{3k} 2^{pi}$ and $T = a([m])$.

Claim: Matchings correspond exactly to subsets $I \subseteq [m]$ such that $a(I) = S$.

To see why, think of a_i as bit pattern that selects exactly one element in X, Y and Z, rather than a numerical value.

Here is the critical trick: by the choice of p, we can recover the bit patterns from a sum: there is no way to fake an entry in some p-block by adding a sufficient number of 1s from the next p-block.

This would be blatantly false if we used, say, $p = 1$.

The End

Lastly, define the two filler elements to be

$$a_\alpha = 2T - S \qquad a_\beta = T + S$$

By the definition of T, we cannot have both a_α and a_β on the same side of any valid partition of $[n]$.

So suppose the partition looks like $I \cup \{\alpha\}$ and $\bar{I} \cup \{\beta\}$ where $I \subseteq [m]$.

Then $a(I) + 2T - S = a(\bar{I}) + T + S$ and therefore
$2a(I) = a(I) + T - a(\bar{I}) = 2S$.

Done.

Pseudo-Polynomial Time

To see that Partition is pseudo-polynomial time, consider an instance $a_1, \ldots, a_n$ and set $2B = a([n])$.

The basic idea is to use standard dynamic programming: we compute a table of all the sums $\{\, a(I) \mid I \subseteq [n] \,\}$ truncated at B.

To this end, define a Boolean matrix $P(k, b)$, $1 \le k \le n$, $0 \le b \le B$, by:

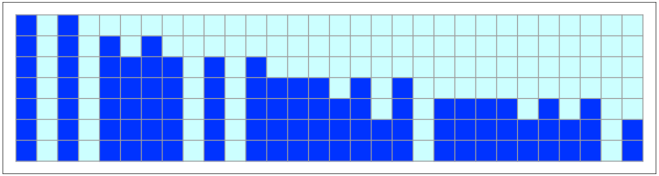
$$P(k, b) \Leftrightarrow \exists\, I \subseteq [k] \left(a(I) = b\right).$$

Clearly we have

$$\begin{aligned} P(1, b) &\Leftrightarrow b = 0 \lor b = a_1 \\ P(k+1, b) &\Leftrightarrow P(k, b) \lor P(k, b - a_{k+1}). \end{aligned}$$

Thus P can be computed in $O(nB)$ steps.

Example



The matrix for $(2, 4, 5, 7, 9, 11, 20)$, $B = 29$.

This is a Yes-instance.

Subset Sum

Problem:	**Subset Sum**
Instance:	A list of natural numbers $a_1, \ldots, a_n, b$.
Question:	Is there a subset $I \subseteq [n]$ such that $\sum_{i \in I} a_i = b$?

Claim

Subset Sum is $\mathbb{NP}$-complete

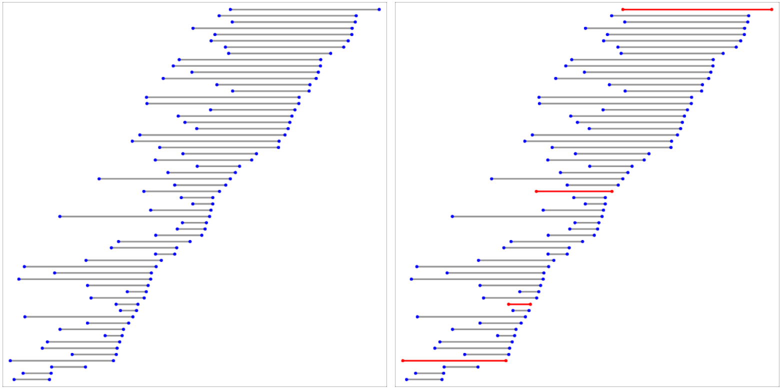
Proof.

Reduction from Partition: keep the a_is, set $b = a([n])/2$. $\square$

Exercise

This a bit terse, explain what's really going on.

Scheduling



Just to be clear: there are many kinds of scheduling problems, some of them have perfectly good polynomial time solutions.

Hard Scheduling

Suppose we have n jobs, each associated with a

- release time r_i
- deadline d_i
- duration Δ_i

all natural numbers.

All jobs must execute on a single processor in *contiguous* time somewhere in the interval $[r_i, d_i]$.

The question is: is there a schedule so all jobs finish by their deadline?

Lemma

Scheduling is $\mathbb{NP}$*-complete.*

Proof

Membership is obvious, for hardness reduce from Subset Sum.

Let $a_1, \dots, a_n$, b be an instance of Subset Sum and let $\beta = a([n])$. Note that we may safely assume $b \leq \beta$. Now define a scheduling instance by

$$r_i = 0 \qquad d_i = \beta + 1 \qquad \Delta_i = a_i$$

plus one additional job $n + 1$ with

$$r_{n+1} = b \qquad d_{n+1} = b + 1 \qquad \Delta_{n+1} = 1$$

Clearly, job $n + 1$ can only run in $[b, b + 1]$.

Since all the jobs must run without gaps, some of the jobs, say $I \subseteq [n]$, must run in $[0, b]$ and thus $a(I) = b$.

The opposite direction is similar.

$\square$

It’s a Trap

Problem:	**Graph Components**
Instance:	A ugraph $G = \langle V, E \rangle$, a number $k \leq |V|$.
Question:	Is there a collection of connected components of G containing k nodes altogether?

Reduction from Subset Sum: build a graph with connected components of size a_i, $i \in [n]$. Set $k = b$.

Exercise

What could possibly go wrong?

Pizza Toppings

5 kinds of pizza toppings: pepperoni, sausage, anchovies, mushrooms, eggplant; to be combined in amounts $x_1, \ldots, x_5$. Each topping has a cost c_j.

Each topping also contains a certain amount of key nutrients, say, carbohydrates, fats, vitamins. Express the content of nutrient i in topping j by coefficient a_{ij}. Lastly, assume that there is a minimal daily allowance b_i for each key nutrient.

$$\begin{aligned} \text{healthy:} \quad & \sum_j a_{ij} x_j \geq b_i \\ \text{cheap:} \quad & \text{minimize } \sum_j c_i x_j . \end{aligned}$$

$\sum_j c_i x_j$ is the objective function

Linear Programming

Expressed in matrix/vector notation A and b:

$$\begin{array}{ccccc|c} 3 & 4 & 1 & 0 & 0 & 5 \\ 4 & 5 & 1 & 0 & 0 & 15 \\ 0 & 0 & 1 & 2 & 3 & 20 \end{array}$$

$c = (4, 3, 5, 2, 1)$.

Has solution $(0, 3, 0, 0, 20/3)$.

Seriously: dozens of implementations, 100s of books, 1000s of papers, dozens of companies, 2 Nobel prizes.

More Formally

An instance of Linear Programming (LP) expresses a minimization problem for n variables and m constraints, with a linear objective function.

More precisely, we have an $m \times n$ integer matrix A, $m \leq n$, a m-component integer vector b and an n-component integer vector c.

One has to find a real vector $x \in \mathbb{R}^n$ that

$$\begin{aligned} \text{minimize } z = c \circ x \\ Ax \geq b \\ x \geq 0 \end{aligned}$$

Note that these are all very natural geometric conditions, using just a bit of linear algebra.

Geometry

For canonical form LP's there is a natural geometric interpretation.

$$F = \{\, x \in \mathbb{R}^n \mid Ax \geq b \wedge x \geq 0 \,\}$$

is a convex polytope in n-dimensional space and contained in the first orthant.

This is called the set of feasible solutions or the simplex.

For any number d the set $\{\, x \in \mathbb{R}^n \mid c \circ x = d \,\}$ is a hyperplane perpendicular to c.

Thus we have to find the first point in F where a hyperplane perpendicular to c intersects F (if it is moved from infinity towards the simplex in the appropriate direction).

Simplex Algorithm

The Simplex algorithm is an iterative procedure that moves from vertex to vertex on the simplex F, decreasing the objective function step by step until a minimum is reached.

Basic Idea: Dantzig 1947

- Find a vertex of the feasible region.
- Consider all immediate neighbors of the current vertex.
- If none of them provide a better value for the objective function, stop.
- Otherwise pick one that does, and repeat.

Running Time

Klee and Minty showed in 1972 how to construct bad inputs that produce exponentially long isotonic paths (objective function increases monotonically along the path). But, depending on the implementation details, Simplex may not choose such a long path.

Also, the counter examples tend to be somewhat artificial. The average running time of Simplex is polynomial, both empirically and theoretically (Smale's proof attracted a huge amount of criticism).

Other Algorithms

In 1979 Khachiyan developed a polynomial time algorithm, the so-called Ellipsoid Algorithm for LP; unfortunately, it seems that the algorithm is not practical.

However, in 1984 Karmarkar found a polynomial time algorithm that seems to be competitive with Simplex on practical inputs, a so-called interior point method.

Expressiveness of LP

To appreciate the power of LP, note that one can easily express rather complicated flow problems as LP.

Variables:	x_e is flow along an edge e.
Constraints:	$0 \leq x_e \leq c(e)$.
Conservation:	$\sum_{e=(u,x)} x_e = \sum_{e=(x,u)} x_e$.
Maximize:	$\sum_{e=(s,x)} x_e - \sum_{e=(x,s)} x_e$

Integer Programming (IP)

It is tempting to ask what happens if we try to solve a Linear Program over $\mathbb{Z}$ rather than $\mathbb{R}$.

As Matiyasevic has shown, solving multivariate polynomial equations over $\mathbb{Z}$ turns out to be hugely more difficult than over $\mathbb{R}$: a (highly nontrivial) decidable problem goes rogue and becomes undecidable.

However, we are saved by the fact that we are dealing with linear algebra here. Note, though, that IP is not obviously in $\mathbb{NP}$: it is not clear that the solution is small–but, again, some linear algebra considerations show that there actually is no problem.

Example: 3SAT as IP

Variables: indicator variable x_v for each Boolean variable x

Constraints: $0 \leq x_v \leq 1$, $1 \leq x'_v + y'_v + z'_v$ for each clause $\{x, y, z\}$

Minimize: $\sum x_v$.

Here $x'_v = x_v$ if x appears positively, $x'_v = 1 - x_v$ otherwise. To get a decision problem, distinguish between feasible and not feasible.

This is an example of 0/1-Integer Programming: the variables are constrained to $\mathbf{2}$ (and membership in $\mathbb{NP}$ is trivial).

Claim

0/1-Integer Programming is $\mathbb{NP}$-complete.