UCT

Karp's List

Klaus Sutner

Carnegie Mellon University

Spring 2022

1. **Karp's List**

2. **More Reductions**

3. **Numerical Problems**

In 1971, Steve Cook (a student of Hao Wang) published his seminal paper on $\mathbb{NP}$-completeness (7945 citations). Levin's paper was not available 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.

Note that, if Cook's paper had been confined to proof theory, no one (except of few logicians) would have cared one bit.

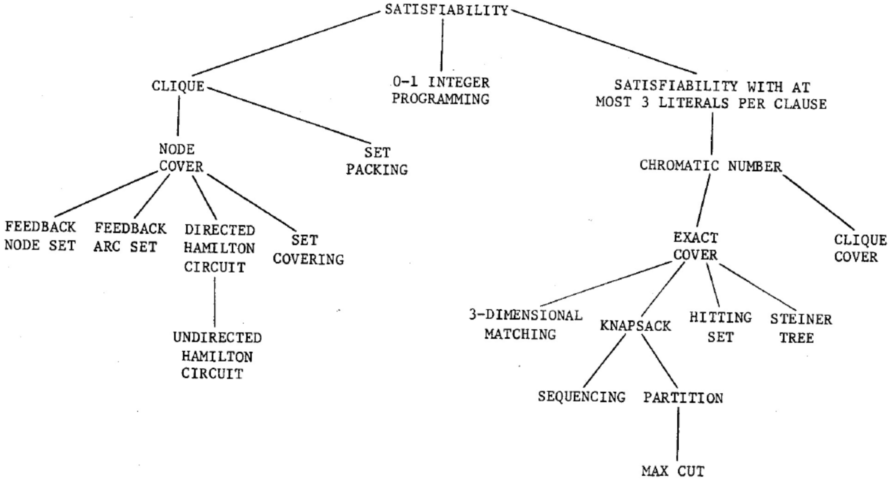
Richard Karp at Berkeley had a habit of reading Cook's papers–and when he saw the SAT paper he realized that this was just the tip of an iceberg.

> 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.

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

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, $|W \cap U_i| = 1$.

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.

- 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 Khachiyan, 1979.

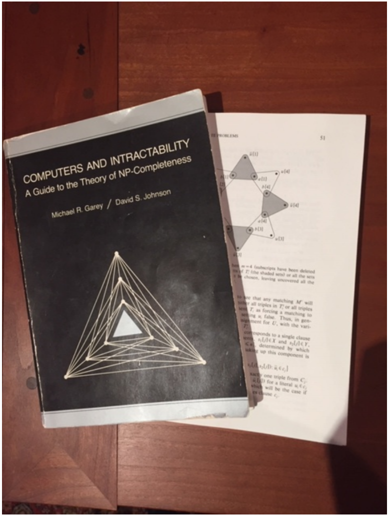
  Similarly, Khachiyan's original method is not practical. However, there are now interier point methods that are polynomial time and are competitive with Dantzig's classical simplex algorithm at least for some instances.

- Graph isomorphism is still a mess (look at Babai's recent work).

  This turns out to be the most intransigent problem. 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 things will go.

- 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.



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

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

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

For a truth assignment σ, define its weight 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 3SAT.

Let n be the number of Boolean variables in the instance of 3SAT 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.

$\square$

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 smallest 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.

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.

$\square$

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).

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.

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 $2mn$.

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 $x_i = X$.

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.

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.

$\square$

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.

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.

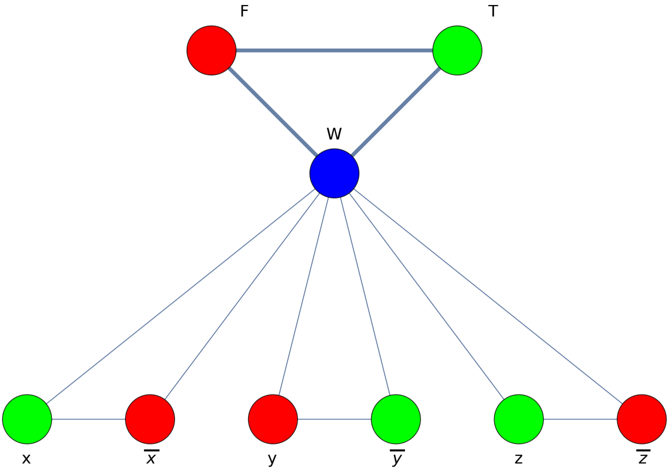
Membership is obvious, for hardness embed 3SAT.

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

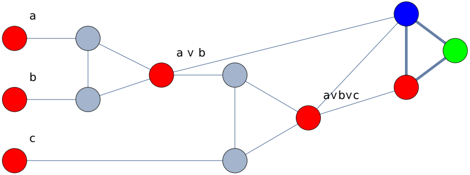
Introduce a triangle with nodes W, T, F (what, true, false).

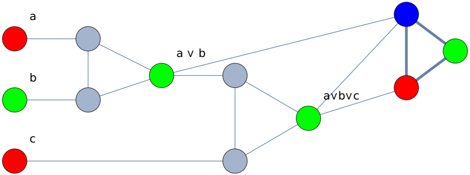
For each variable x, there is a truth setting edge $\{x, \overline{x}\}$ and both nodes are connected to W.

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.


F
T
W
x
$\overline{x}$
y
$\overline{y}$
z
$\overline{z}$

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.

□

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.

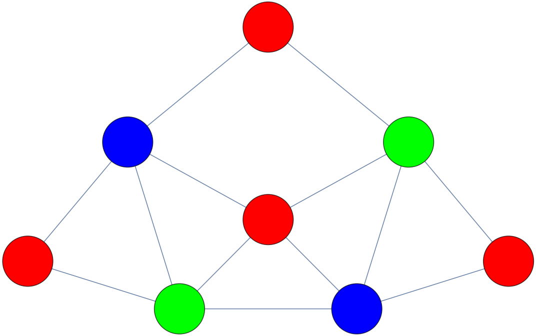
Lemma

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.

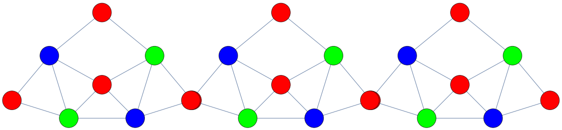
H_k will have degree 4, and will have k terminal nodes to which we can connect the neighbors of the high degree vertex.



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

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.

$\square$

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).

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

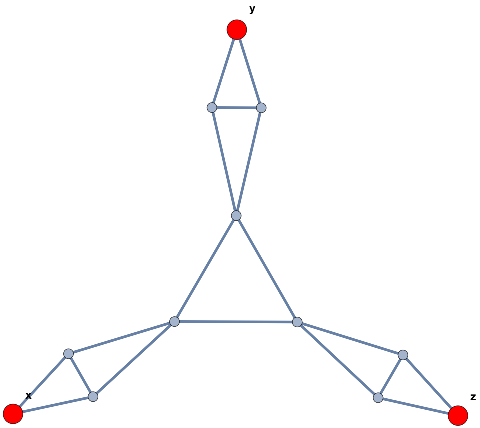
$$(x_i, a_i^1), (x_i, a_i^2), (a_i^1, a_i^2), (a_i^1, a_i^3), (a_i^2, a_i^3)$$

and likewise for y–b and z–c.

Lastly, add a central triangle (see pic)

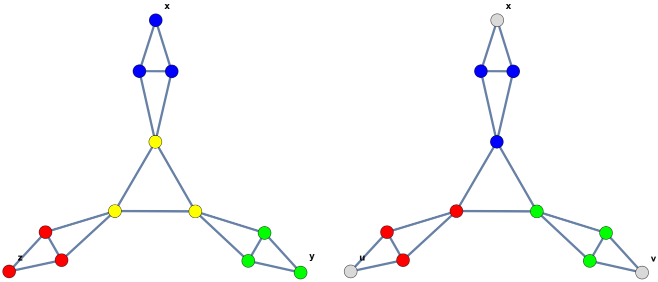
$$(a_i^3, b_i^3), (a_i^3, c_i^3), (b_i^3, c_i^3)$$

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



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

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.

□

Theorem

Hamiltonian Cycle is $\mathbb{NP}$*-complete.*

Note that this is a clean decision problem, taken directly from graph theory. There is no artificial bound to force things into this format.

A similar problem is Hamiltonian Path: we are looking for a path that touches every vertex exactly once (but need not form a cycle).

Membership is obvious, for hardness reduce from Vertex Cover.

Let $G = \langle V, E \rangle$ be a ugraph, and k a bound, $1 \leq k \leq n = |V|$. We may assume wlog that all vertices have degree at least 2 (why?).

For each v, fix an enumeration u_i^v, $i \in [\deg(v)]$, of all its neighbors.

Define a new graph H as follows:

Vertices

- anchor vertices $a_1, \ldots, a_k$
- box vertices $\boxed{e, v, i}$ for all $v \in e \in E$, $i \in [6]$.

Edges

- chain edges (connecting anchors and boxes)
- box edges (inside a box)

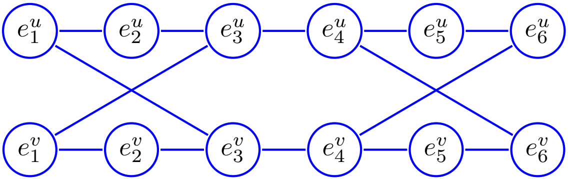
Chain Edges:

- $\{a_j, \boxed{e, u_1^v, 1}\}$ and $\{a_j, \boxed{e, u_{\deg(v)}^v, 6}\}$ for all $j \in [k]$.
  These edges connect the anchor points to the first and last box on the v-chain. e is understood to be $\{v, u_i^v\}$.
- $\{\boxed{e, u_i^v, 6}, \boxed{e, u_{i+1}^v, 1}\}$ for $1 \leq i < \deg(v)$,
  These edges connect two consecutive boxes on the v-chain.

Box Edges:

$\{\boxed{e, v, i}, \boxed{e, v, i+1}\}$, $\{\boxed{e, v, 1}, \boxed{e, u, 3}\}$ and $\{\boxed{e, v, 4}, \boxed{e, u, 6}\}$ for all $\{u, v\} = e \in E$.

These edges form a 12-point gadget, a box that appears on the v-chain.



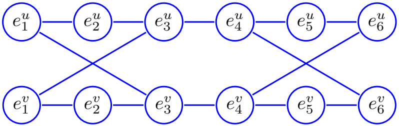
The box representing edge $e = \{u, v\}$.

It is connected to the rest of the graph only at the 4 corners (to form a chain, and connect to the anchor vertices).

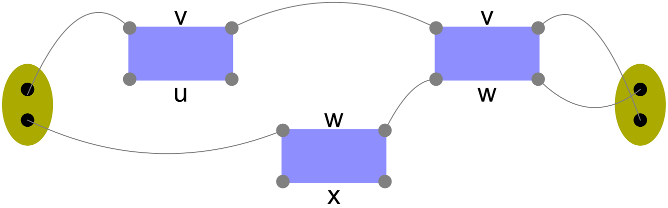
Now assume that P is a Hamiltonian cycle in G.

Claim 1: P must enter and exit each box at the same side. P can pass through the e-box in exactly one of two ways: type full (covers all vertices) or type half (covers only the points on the side where it entered).

To see this, take a pen and try to traverse the box. There simply are no other possibilities.



u — v — w — x



Claim 2: Without loss of generality, P consists of blocks

$$a_i, \boxed{e, u_1^v, 1}, \ldots, \boxed{e, u_{d(v)}^v, 6}, a_{i+1}$$

going from an anchor point to another, and containing the whole v-chain.

Claim 3: G has a vertex cover of size k.

To see this define $C = \{\, v \in V \mid P \text{ uses the } v\text{-chain } \}$. Now let $e = \{u, v\} \in E$. As P passes through the e-box it must use the u-chain or v-chain. Thus C covers e.

Suppose G has a vertex cover of size k.

Claim 4: Then H has a Hamiltonian cycle.

Construct P as follows: P has k blocks $a_i, \boxed{e, u_1^v, 1}, \ldots, \boxed{e, u_{d(v)}^v, 6}, a_{i+1}$ where v is in C.

The way P passes through the e-box on the v-chain is determined by whether $u \in C \wedge v \in C$ (type half) or $u \in C \oplus v \in C$ (type full).

Thus G has a VC of size k iff H is Hamiltonian. Clearly, G can be constructed in polynomial time.

$\square$

Corollary

Traveling Salesman is $\mathbb{NP}$-complete.

Corollary

Hamiltonian Path is $\mathbb{NP}$-complete.

Corollary

Longest Path is $\mathbb{NP}$-complete.

Exercise

Verify all these claims.

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?

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.

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.

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.

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.

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:

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

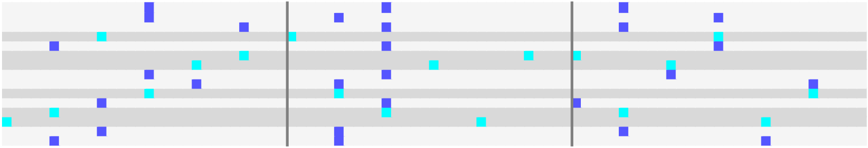
The index set for Partition is $[n] = [m] \cup \{\alpha, \beta\}$. To describe the integers a_i, let $p = \lceil \lg m \rceil + 1$ and set for $i \in [k]$;

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

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.

	...	1000	...		‖		...	1000	...		‖		...	1000	...	

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.



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.

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.

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 1s from the next p-block.

Lastly, define the two filler elements to be

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

Now suppose we have $I \subseteq [n]$ such that $a(I) = a([n] - I)$. Wlog $\alpha \in I$ and $\beta \notin I$. But then $a(I - \{\alpha\}) = S$, and we are done.

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 compute all the sums $\{\, a(I) \mid I \subseteq [n] \,\}$ truncated at B.

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

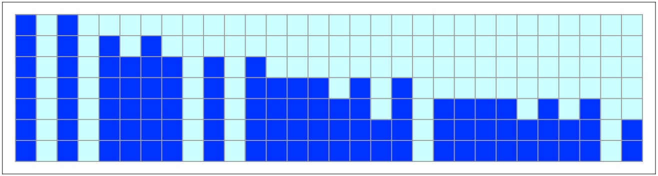
$$P(k, b) \Leftrightarrow \exists\, I \subseteq [k] \left(a(I) = b\right).$$

Clearly we have

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

This is just standard dynamic programming. Thus P can be computed in $O(nB)$ steps.

Example



The matrix for $(2, 4, 5, 7, 9, 11, 20)$, a Yes-instance.

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

Claim

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

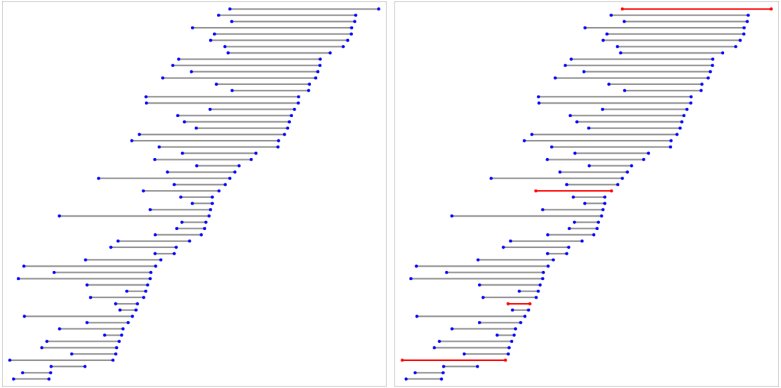
Proof.

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

Exercise

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

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



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 $[r_i, d_i - \Delta_i]$.

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

Lemma

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

Membership is obvious, for hardness reduce from Subset Sum.

Let $a_1, \ldots, a_n$, b be an instance of Subset Sum, set $\beta = a([n])$ and

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

Note that we may safely assume $b \leq \beta$.

Now introduce a new 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.

□

Problem:	**Graph Components**
Instance:	A ugraph $G = \langle V, E \rangle$, a number $k \leq \lvert V \rvert$.
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?