Intermediate Problems

Klaus Sutner

Carnegie Mellon University

1 **Intermediate Problems**

2 **Graph Isomorphism**

Complexity Classes

A complexity class $\mathcal{C}$ is a collection of decision problems that are all defined by a common resource bound.

Useful classes have closure properties:

Intersection Logical "and", sequential composition.

Union Logical "or", parallel composition.

By contrast, closure under complement often fails, or is not known.

We will ignore other complexity classes based on function problems, counting problems, optimization problems, etc.

Four Examples

- Semidecidable, Σ_1
- Decidable, Δ_1.
- Nondeterministic polynomial time, $\mathbb{NP}$.
- Deterministic polynomial time, $\mathbb{P}$.

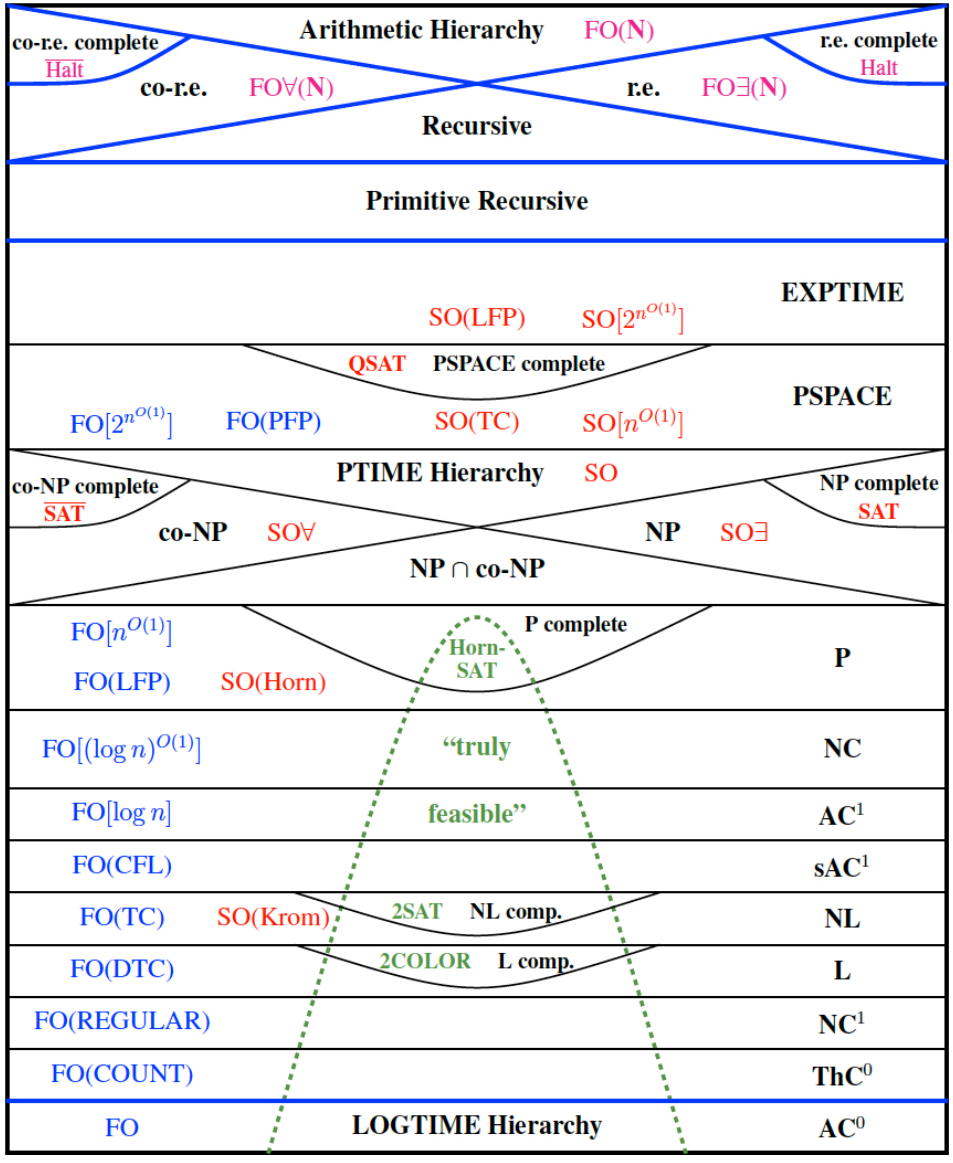
Clearly

$$\mathbb{P} \subseteq \mathbb{NP} \subseteq \Delta_1 \subseteq \Sigma_1$$

All inclusions except the first are known to be proper.

Formally we can always think of $\mathcal{C} \subseteq \mathfrak{P}(\mathbf{2}^\star)$: code the Yes-instances of a problem as a formal language.

A Zoo


Arithmetic Hierarchy
FO(N)
co-r.e. complete
Halt
co-r.e.
FO∀(N)
r.e. complete
Halt
r.e.
FO∃(N)
Recursive
Primitive Recursive
EXPTIME
SO(LFP)
SO[2^{n^{O(1)}}]
QSAT
PSPACE complete
PSPACE
FO[2^{n^{O(1)}}]
FO(PFP)
SO(TC)
SO[n^{O(1)}]
PTIME Hierarchy
SO
co-NP complete
SAT
co-NP
SO∀
NP
SO∃
NP complete
SAT
NP ∩ co-NP
FO[n^{O(1)}]
FO(LFP)
SO(Horn)
Horn-SAT
P complete
P
FO[(log n)^{O(1)}]
"truly
NC
FO[log n]
feasible"
AC^1
FO(CFL)
sAC^1
FO(TC)
SO(Krom)
2SAT
NL comp.
NL
FO(DTC)
2COLOR
L comp.
L
FO(REGULAR)
NC^1
FO(COUNT)
ThC^0
FO
LOGTIME Hierarchy
AC^0

Reductions

A complexity class $\mathcal{C}$ typically comes equipped with a reduction relation, a pre-order (reflexive and transitive) $\leq$ that corresponds to "easier than:" for $A, B \in \mathcal{C}$ we have $B \leq A$ if a decision algorithm for A gives rise to a decision algorithm for B (using only limited resources).

Example: For semidecidable sets $A \leq B$ means that membership in A can be determined with B as oracle (no bounds on the computation).

Example: For $\mathbb{NP}$ sets $A \leq B$ means that there is a polynomial time computable function $f : \mathbf{2}^\star \to \mathbf{2}^\star$ such that

$$x \in A \iff f(x) \in B$$

These are the standard reductions, but there are several other possibilities.

Hardness and Completeness

Suppose $\mathcal{C}$ is a complexity class $\mathcal{C}$ with reduction $\leq$.

A is $\mathcal{C}$-hard if for all $B \in \mathcal{C}$: $B \leq A$.

A is $\mathcal{C}$-complete if A is $\mathcal{C}$-hard and $A \in \mathcal{C}$.

Thus, the $\mathcal{C}$-complete problems are the most difficult problems in $\mathcal{C}$ (wrt the reduction $\leq$).

The Halting set H is the standard example of a complete semidecidable set.

Likewise, Satisfiability is complete for $\mathbb{NP}$.

Degrees

The reduction relation $\leq$ on $\mathcal{C}$ is a pre-order (but typically not total). Hence it induces an equivalence relation

$$A \equiv B \iff A \leq B \text{ and } B \leq A$$

The equivalence classes are called degrees: all problems in a degree are of similar difficulty.

We write $\deg A$ for the degree of $A \subseteq \Sigma^\star$.

Turing Degrees

In general computability, the most important example of a reduction is Turing reducibility: $A \leq_T B$ if there is a Turing machine with oracle B that decides A.

The corresponding degrees are called Turing degrees or degrees of unsolvability, written $\deg_T A$.

There are two complexity classes for which Turing reducibility is natural:

- all subsets of $\mathbb{N}$, written $\mathcal{D}$
- all semidecidable subsets of $\mathbb{N}$, written $\mathcal{E}$.

These have been studied to death, an beyond; see below.

Upper Bounds

Given $A, B \subseteq \mathbb{N}$, define

$$A \oplus B = (2A) \cup (2B + 1)$$

Claim: The degree of $A \oplus B$ is the least degree above $\deg_T A$ and $\deg_T B$.

If this sounds lame, try to do something similar for below, some kind of meet rather than join.

The Monster $\mathcal{D}$

Since $\mathcal{D}$ is based on the full powerset of $\mathbb{N}$ it is uncountable.

But note that all Turing degrees are countable, so there must be uncountably many degrees. Also, each degree has only countably many predecessors.

The following are quite difficult to prove and rather counter-intuitive.

Theorem

There is an uncountable set of degrees that are pairwise incomparable.

Theorem

Every countable partial order can be embedded into $\mathcal{D}$ as an initial segment.

Easy Problems

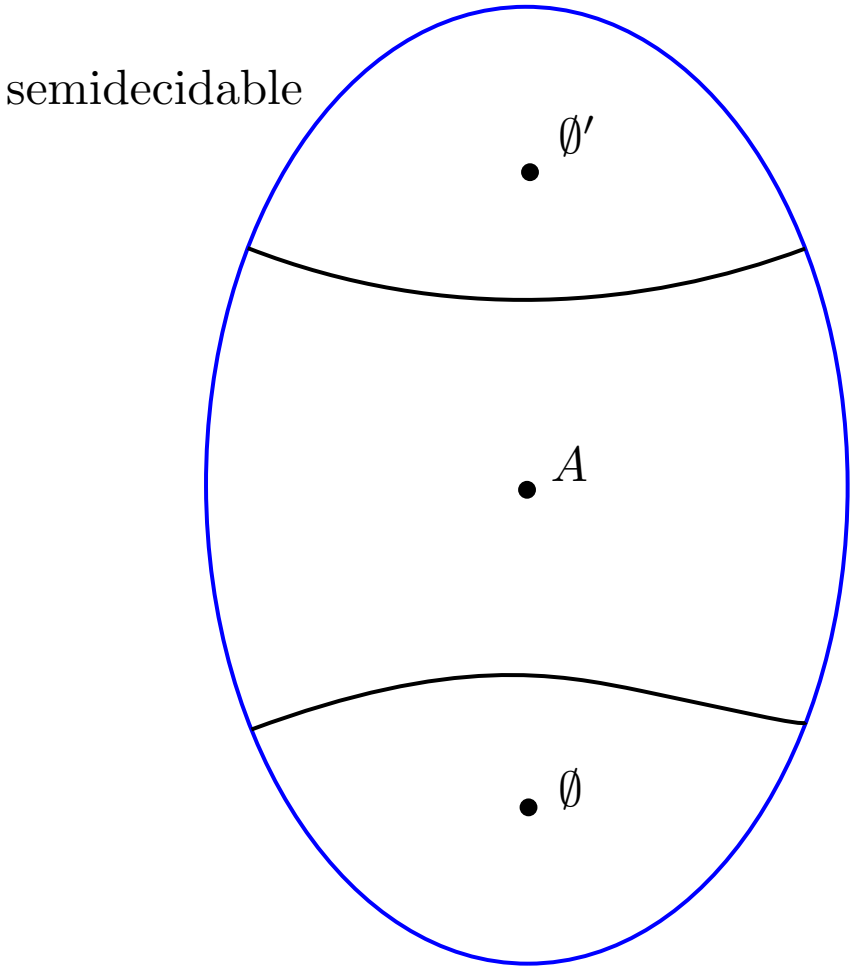
Often a complexity class $\mathcal{C}$ also has a subclass $\mathcal{C}^- \subseteq \mathcal{C}$ that consists of all problems in $\mathcal{C}$ that are "easy" in a certain sense.

Example 1: decidable is easy within $\mathcal{E}$, the class of semidecidable problems.

Example 2: $\mathbb{P}$ is easy within the class of $\mathbb{NP}$-problems.

Note one huge difference between the two examples: we can prove that $\mathcal{C}^- \neq \mathcal{C}$ in the first case, but not in the second. And it may even be the case that $\mathbb{P} = \mathbb{NP}$.

Post's Question



Is this picture of $\mathcal{E}$ accurate?

Is there a semidecidable set A such that $\emptyset < A < H$?

Why?

Because it is a matter of experience (not theory) that any problem that is recognized as being semidecidable ultimately will either

- turn out to be complete, or
- turn out to be decidable.

In other other words, semidecidable sets that occur in the RealWorld$^{\mathsf{TM}}$ either belong to the degree of $\emptyset$ or the degree of $\emptyset'$.

It may take a long time to determine which is the case (Diophantine equations took 70 years), but in the end everything turns out to be decidable or complete. Basta.

Emil Post and Friedberg/Muchnik

So E. Post asked in 1944 whether there are any intermediate semidecidable sets (i.e., more than just two semidecidable degrees).

Surprisingly, two people found the solution almost simultaneously: R. M. Friedberg (an undergraduate) in the US and A. A. Muchnik in Russia (they published their results in 1957 and 1956, respectively).

Theorem (Friedberg, Muchnik 1956/7)

There are intermediate semidecidable problems.

What is even more surprising, they used essentially the same method (a so-called finite injury priority method). We'll sketch the argument below, but a complete proof is too much work.

How to Construct a Semidecidable Sets

To describe the construction it is best to use the characterization of semidecidable sets as being recursively enumerable.

The construction proceeds in stages $\sigma \in \mathbb{N}$.

At stage σ one builds a finite set $A_\sigma \subseteq \mathbb{N}$. In the end we set

$$A := \bigcup_{\sigma \geq 0} A_\sigma$$

More precisely, there is a primitive recursive function C, the construction, such that

$$A_\sigma = C(\sigma, A_{<\sigma})$$

where $A_{<\sigma} = \bigcup_{\tau<\sigma} A_\tau$.

Note that once an element has been added to A, it cannot be taken out at a later stage.

A Trick

It suffices to construct two incomparable semidecidable sets, i.e., two sets A and B such that

$$A \not\leq B \qquad \text{and} \qquad B \not\leq A$$

Right?

The two sets are constructed in parallel in stages. To make sure they have the right properties we use requirements:

(R_e) $A \neq \{e\}^B$ insure $A \not\leq B$

(R'_e) $B \neq \{e\}^A$ insure $B \not\leq A$

Organizing Work

Note that there are infinitely many requirements, two for each index $e \in \mathbb{N}$. So we have to be careful about organizing our construction.

- At stage σ we only consider objects $< \sigma$ (think about hereditarily finite sets, truncated at σ).
- In particular we only consider requirements (R_e) and (R'_e) for $e < \sigma$.
- And we run computations only for at most σ steps.

As a result, the construction at stage σ is really just primitive recursive.

And the sets A and B are indeed semidecidable.

But How?

Suppose we find at stage σ that requirement (R_e) is currently broken:

$$A_{<\sigma} = \{e\}^{B<\sigma}$$

Bear in mind our restriction to a world below σ, this involves only finitely many numbers.

Suppose also that we have some witness x for (R_e):

$$x \notin A_{<\sigma} \qquad \text{and} \qquad \{e\}^{B<\sigma}(x) = 0$$

Then we throw x into A_σ: requirement (R_e) requires attention and receives attention. For the time being, the requirement is met and goes to sleep.

OK, so we have just satisfied requirement (R_e).

But there are other requirements, in particular (R'_i):

$$B \neq \{i\}^A$$

Since we just changed A, it may happen that some computations using oracle A now change, and so now we have $B = \{i\}^A$ up to σ.

Requirement (R_e) has just injured requirement (R'_i).

And, of course, if we fix (R'_i) we may clobber some other requirement, and so on. We may satisfy a few requirements, but there apparently is no way to deal with all infinitely many of them.

Priority

Here is the central idea: we prioritize requirements as in

$$R_0 > R_0' > R_1 > R_1' > R_2 > R_2' > \dots$$

We will work on (R_e) at even stages, on (R_e') at odd stages.

More importantly, a requirement can only receive attention (throw some witness x into A or B) if that does not injure a higher priority requirement.

This has the effect that the high priority requirements get taken care of first, then the lower priority ones. In the limit, all requirements are satisfied (proof is by induction).

Density

Friedberg/Muchnik shows that $\mathcal{E}$ has incomparable and thus intermediate elements. Here is another strange result, due to Sacks.

Theorem (Density)

Suppose A and B are semidecidable and $A <_T B$. There there is another semidecidable set C such that $A <_T C <_T B$.

Repeating this construction we can generate a dense order of semidecidable sets. This is not particularly intuitive, to say the least.

Natural Intermediate Problems?

Alas, the Friedberg/Muchnik construction is really quite frustrating: it builds a semidecidable set that is undecidable and strictly easier than Halting–but it has no purpose other than this; it is totally artificial.

So it is tempting to ask whether there are natural intermediate problems.

For example, maybe there is some reasonable class of integer polynomials for which the question of the existence of integral roots is intermediate.

Not a single example of such a natural intermediate problem is known today. All natural semidecidable problems have ultimately turned out to be either decidable, or complete (sometimes requiring huge effort). A lot of people find that rather frustrating.

70 years of research on Turing degrees has shown the structure to be extremely complicated. In other words, the hierarchy of oracles is worse than any political system. No one oracle is all powerful.

Suppose some quantum genius gave you an oracle as a black box. No finite amount of observation would tell you what it does and why it is non-recursive. Hence, there would be no way to write an algorithm to solve an understandable problem you couldn't solve before! Interpretation of oracular statements is a very fine art–as they found out at Delphi.

The heavy symbolism used in the theory of recursive functions has perhaps succeeded in alienating some mathematicians from this field, and also in making mathematicians who are in this field too embroiled in the details of their notation to form as clear an overall picture of their as is desirable. In particular the study of degrees of recursive unsolvability by Kleene, Post, and their successors has suffered greatly from this defect, . . .

The study of degrees seems to be appealing only to some special kind of temperament since the results seem to go into many different directions. Methods of proof are emphasized to the extent that the main interest in this area is said to be not so much the conclusions proved as the elaborate methods of proof.

. . . but one can be quite precise in stating that no one has produced an intermediate r.e. degree about which it can be said that it is the degree of a decision problem that had been previously studied and named.

Formerly, when one invented a new function, it was to further some practical purpose; today one invents them in order to make incorrect the reasoning of our fathers, and nothing more will ever be accomplished by these inventions.

The theory of computation has traditionally been studied almost entirely in the abstract, as a topic in pure mathematics. This is to miss the point of it. Computers are physical objects, and computations are physical processes. What computers can or cannot compute is determined by the laws of physics alone, and not by pure mathematics.

Information is not a disembodied abstract entity; it is always tied to a physical representation. It is represented by an engraving on a stone tablet, a spin, a charge, a hole in a punched card, a mark on paper, or some other equivalent. This ties the handling of information to all the possibilities and restrictions of our real physical world, its laws of physics and its storehouse of available parts.

Principle of Computational Equivalence

Stephen Wolfram in 2002:

> *. . . all processes, whether they are produced by human effort or occur spontaneously in nature, can be viewed as computations.*
>
> *. . . almost all processes that are not obviously simple can be viewed as computations of equivalent sophistication.*

Clearly false in the setting of classical computability theory. But if one works in a physics-like model of computation, it might actually work out.

How About $\mathbb{P}$ vs. $\mathbb{NP}$?

Here we don't even know that the classes are distinct (and some notable researchers like Levin and Knuth claim they are not).

In 1972 Karp identified 3 potentially intermediate problems for $\mathbb{NP}$:

- Graph Isomorphism
- Nonprime
- Linear Inequalities

The Primality Shock

Pratt has shown that Primality is in $\mathbb{NP}$ (a beautiful application of number theory). So both Primality and Nonprime are in $\mathbb{NP} \cap$ co-$\mathbb{NP}$, and thus unlikely to be $\mathbb{NP}$-complete. Sure enough ...

Theorem (Agrawal-Kayal-Saxena 2002)

Primality is in polynomial time.

Annoyingly, their algorithm uses no more than high-school algebra.

AKS does not seem to provide a computationally superior method, probabilistic algorithms are far better.

Linear Inequalities

In Karp's version one has to decide if there exists a rational vector $\boldsymbol{x}$ such that

$$M \cdot \boldsymbol{x} \geq \boldsymbol{a}$$

where M and $\boldsymbol{a}$ are integral.

This comes down to linear programming.

Theorem (Khachiyan 1979)

Linear Programming is in polynomial time.

The proof is quite difficult and does not directly yield superior algorithms (superior to the standard simplex algorithm).

Ladner's Theorem

Theorem (Ladner 1975)

If $\mathbb{P} \neq \mathbb{NP}$, then there are intermediate problems wrto polynomial time reducibility.

The proof is quite similar to the Friedberg/Muchnik construction and produces an entirely artificial example of an intermediate problem.

Since separating $\mathbb{P}$ from $\mathbb{NP}$ is probably rather difficult it currently looks quite hopeless to find natural examples.

The Problem

Problem:	**Graph Isomorphism (GI)**
Instance:	Two undirected graphs G and H.
Question:	Are G and H isomorphic?

GI is trivially in $\mathbb{NP}$: guess the bijection $f : V_G \to V_H$ and verify that it is indeed an isomorphism.

But, GI seems too squishy to support any hardness results, none of the known constructions seems to apply.

Contra Completeness

Many $\mathbb{NP}$-complete problems are fairly robust: one can constrain the instances, but completeness persists.

On the other hand, GI is known to be in polynomial time when

- degrees are bounded,
- genus is bounded,
- tree-width is bounded.

Moreover, if GI were $\mathbb{NP}$-complete, then the polynomial hierarchy would collapse (something most researchers consider unlikely).

Write $\mathcal{G} = \mathcal{G}_n$ for the class of all ugraphs on $[n]$.

An isomorph of G is any graph $H \in \mathcal{G}$ isomorphic to G.

In other words, we need a permutation σ of $[n]$ such that

$$\{i, j\} \in E_G \iff \{\sigma(i), \sigma(j)\} \in E_H$$

One usually writes G^σ for the graph obtained by applying σ.

So G^σ is just a relabeling of G: we give a different name to each vertex.

Total Recall

This is really a right action

$$\mathcal{G} \times \mathfrak{S} \to \mathcal{G}$$

where $\mathfrak{S}$ is the symmetric group on n points. We relabel, but the topology of the graph is unchanged.

Given two graphs G and H we want to know whether $H = G^\sigma$ for some $\sigma \in \mathfrak{S}$. Note that we can safely assume that both graphs are connected.

Of course, brute-force is out, $\mathfrak{S}$ has size exponential size. We need shortcuts that limit the search space.

Degrees

We have already taken care of $|V_G| = |V_H|$ by considering only $\mathcal{G} = \mathcal{G}_n$. We also need $|E_G| = |E_H|$, but we can do slightly better:

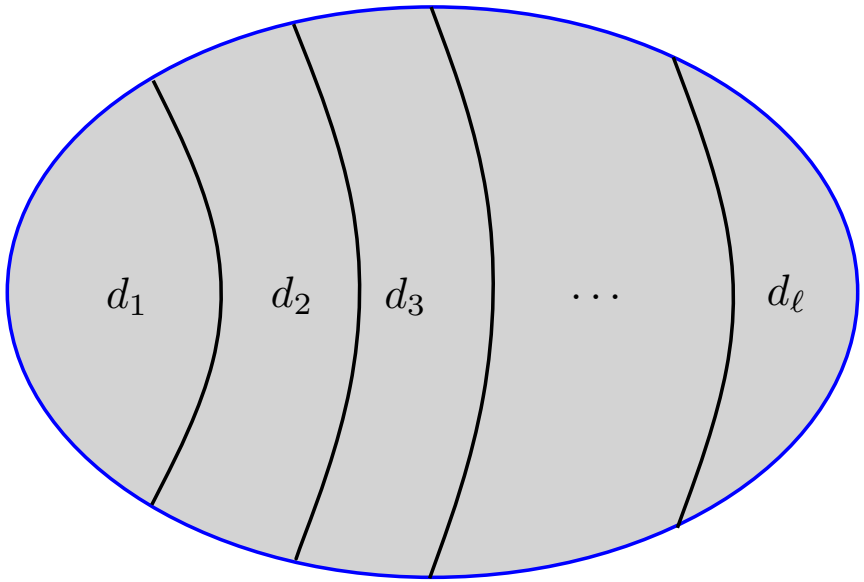
Define the degree of a vertex x as

$$\deg(x) = \text{number of vertices adjacent to } x$$

Then we can partition the vertex sets into blocks by collecting points of the same degree.

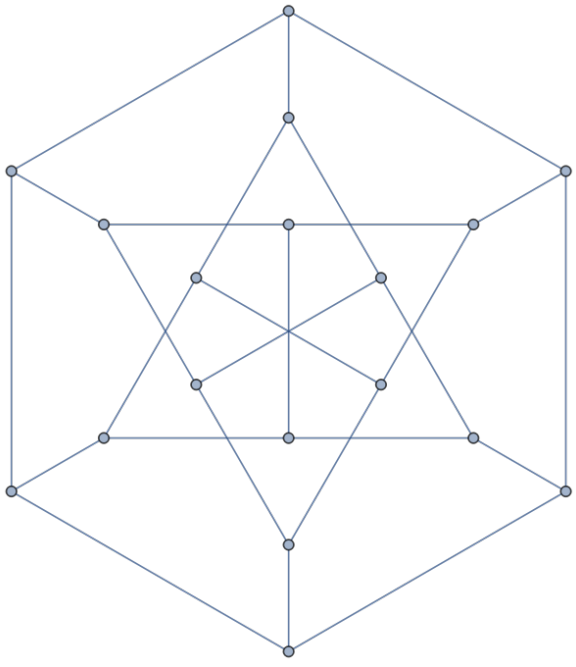
Obviously, these blocks must have the same cardinalities in G and H, so in particular the edge sets must have the same cardinalities.

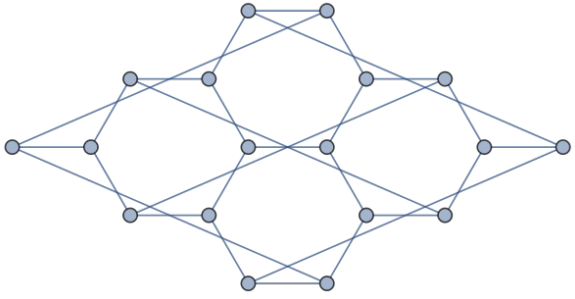
Degree Partition

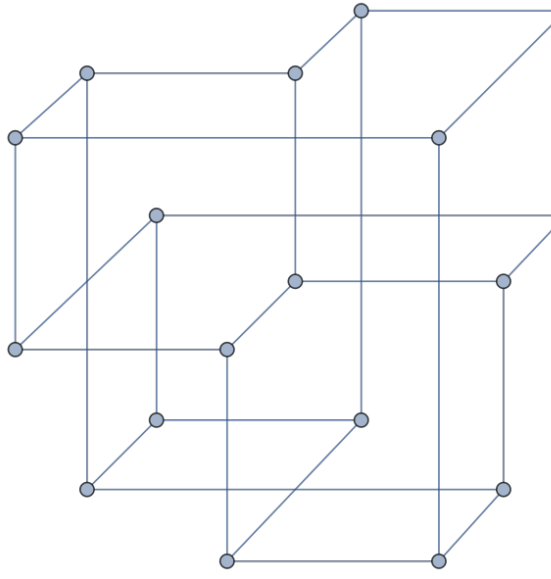


Any isomorphism can then be broken up into smaller maps, one each for vertex of degree d_i.

Three Views of Pappus

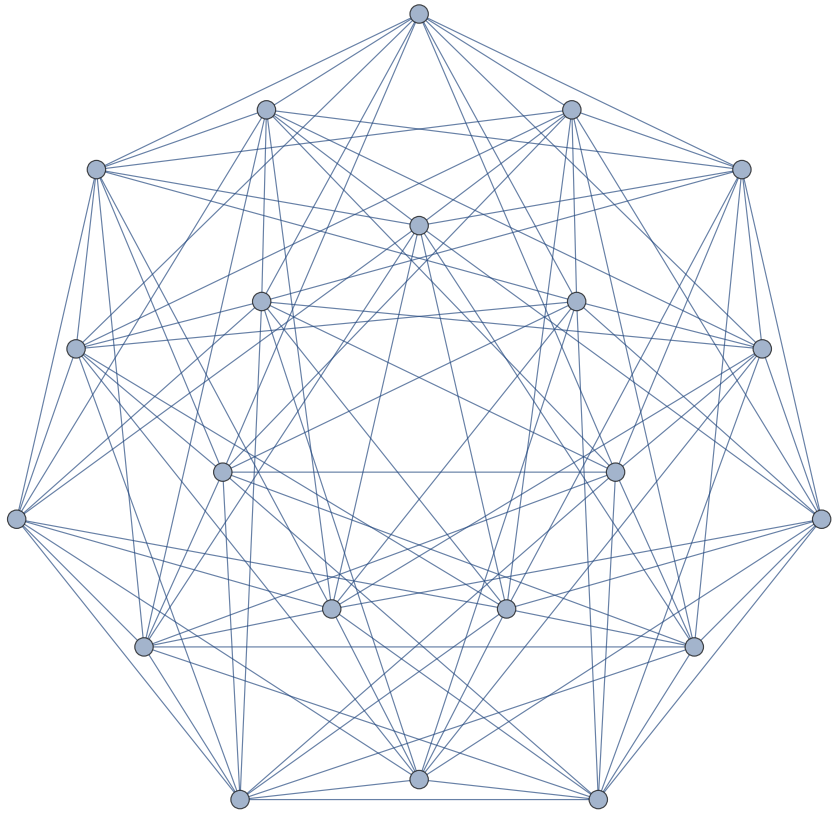






Unfortunately this does not work at all for regular graphs (all vertices have same degree).

Johnson Graphs



A particularly problematic regular graph: a Johnson graph $J(n, t)$.
Vertices are $\mathfrak{P}_t([n])$ and x and y are adjacent if $|x \Delta y| = 2$.

There is an excellent algorithm from the 1980s that usually works well, even on thousands of vertices, but does blow-up exponentially on occasion.

B. D. McKay, A. Piperno

Practical Graph Isomorphism II

J. Symbolic Computation, 60 (2014) 94–112

So, it would not be a complete shock if GI ultimately wound up in $\mathbb{P}$.

Here is the GI algorithm.

Canonical Labelings

The main idea is simple: concoct a reasonably easily computable function

$$C : \mathcal{G} \to \mathcal{G}$$

such that $C(G)$ is an isomorph of G and

$$G \cong H \quad \Longrightarrow \quad C(G) = C(H)$$

One way of thinking about this that we need to relabel the vertices of G in a canonical way, based only on G.

If we similarly relabel any isomorph H of G we wind up with the same graph.

Bit-Vectors

Here is a simple idea that will be useful later.

We can think of $G \in \mathcal{G}$ as a subset of $\mathfrak{P}_2([n])$, and thus as a bit-vector $\beta(G)$ of length $\binom{n}{2}$.

12	13	14	23	24	34
0	0	1	1	1	0

Hence we could define $C(G)$ to be the isomorph H such that $\beta(H)$ is maximal wrto lexicographic order. Alas, this particular relabeling seems to be quite inefficient.

Splitting Blocks

Suppose we have an ordered partition $P = (B_1, B_2, \ldots, B_r)$ of $[n]$.

The algorithm will start at the unit partition $P = ([n])$.

The goal is to produce a discrete partition which has only trivial blocks of size 1.

Block B requires attention if it contains points u and v such that

$$\deg(u, B') \neq \deg(v, B')$$

where B' is some block in P (possibly the same as B). Here $\deg(v, B')$ is the number of nodes adjacent to v in B'.

When a block requires attention we split it into subblocks $X_1, \ldots, X_k$ which are sorted according to $\deg(v, B')$ and spliced into the partition, replacing B.

Equitable Partitions

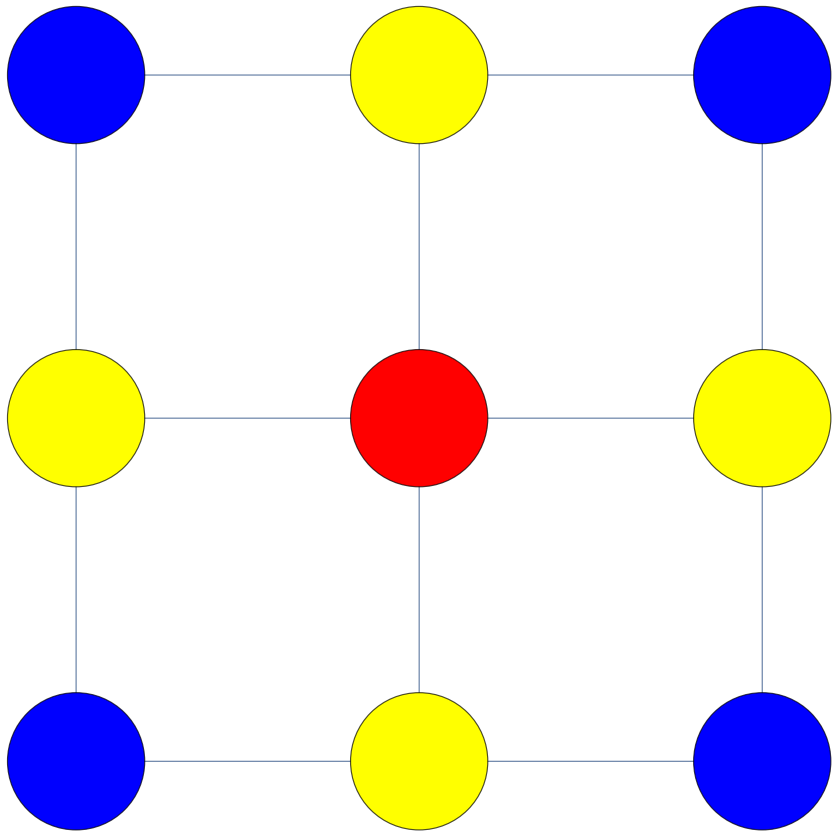
A partition $P = (B_1, B_2, \ldots, B_r)$ is equitable if none of its blocks requires attention.

There is a natural iterative algorithm that computes on input P the coarsest equitable refinement $\eta(P)$ of P: keep splitting until no block requires attention any more. We can even make this deterministic by always picking the first (B, B') pair.

For example, if we start with the unit partition, $\eta([n])$ only contains blocks of vertices with the same degree (but possibly a much finer classification).

This is quite similar to some fast minimization algorithms for DFAs.

Equitable Example



Breaking Symmetry

Alas, an equitable partition will generally not be discrete.

In this case we pick a non-trivial block B and a vertex $u \in B$ and individualize u: we break B into u and $B - u$.

The bad news: since all vertices in B look the same to us, we have to do this for all u in B. The algorithm branches and the branching factor may be $O(n)$.

Then we apply η again, individualize, apply η, individualize and so on.

Partition Tree

This produces a tree $\mathcal{T} = \mathcal{T}(G)$:

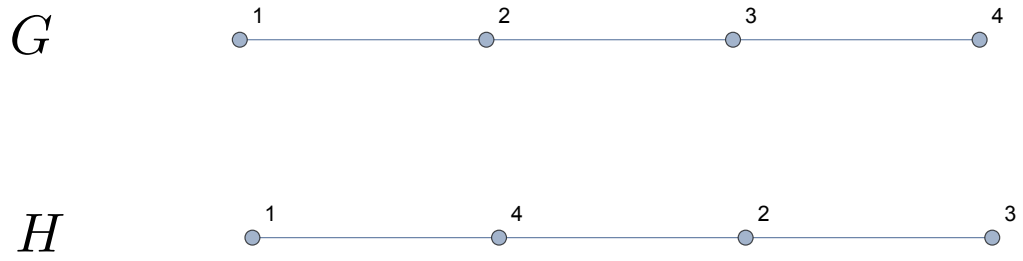
- All nodes are labeled by equitable partitions.
- The root is labeled by $\eta([n])$.
- At internal nodes, all vertices in a block are individualized.
- The leaves are labeled by discrete partitions.

Any discrete, ordered partition $P = (\{a_1\}, \{a_2\}, \ldots, \{a_n\})$ gives rise to a permutation $\sigma = \sigma(P) = (a_1, a_2, \ldots, a_n)^{-1}$.

We take the isomorph with the largest bit-vector as the canonical one:

$$C(G) = H \iff H = G^\sigma \text{ and } \beta(H) \text{ maximal}$$

Pathetic Example



graph	perms σ	bit-vecs
G	(1 4 3 2), (2 3 4 1)	0 0 1 1 0 1
H	(1 3 2 4), (2 4 1 3)	0 0 1 1 0 1

In this particular case, all permutations generate the same bit-vector.

Pruning the Tree

In general, the tree $\mathcal{T}$ will be quite large thanks to blind individualization steps. We need to prune the tree.

Suppose σ and τ are two leaf permutations such that $G^\sigma = G^\tau$. Then

$$\sigma\tau^{-1} \in \mathsf{Aut}(G)$$

Then one can truncate the DFS tree at the lowest common ancestor of the two leaves. Here is where group theory enters the picture.

More on this and Babai's algorithm next week.

Babai's Theorem

Theorem (Babai 2015)

There is a quasipolynomial time algorithm for Graph Isomorphism.

Quasipolynomial means: $2^{O(\log^k n)}$ for some constant k.

So for $k = 1$ we would get polynomial time.

Alas, the algorithm piles up the log factors and we have $\log n$ terms in the exponent.

String Isomorphism

For any set X and alphabet Σ, refer to $\Sigma^X = X \to \Sigma$ as the collection of X-strings over Σ. If X is a finite linear order that's the usual notion.

Note that $\mathfrak{S}_X$ acts naturally on Σ^X (left action) via

$$(\sigma x)(i) = x(\sigma(i))$$

Problem:	**String Isomorphism (SI)**
Instance:	Two strings x and y in Σ^X, a group G acting on X.
Question:	Is $y \in Gx$?

Here G is supposed to be given by a some generators (elements of $\mathfrak{S}_X$). We will use this convention from now on.

Key Theorem

Theorem (Babai)

Given a group G and an X-string x, one can compute the stabilizer G_x in quasipolynomial time.

Again, this means that one can compute a set of generators for G_x, not the actual group.

The idea to use group theory in an attempt to cope with GI goes back to a paper by E. Luks from the 1980s (where he shows that GI is polynomial time for bounded degree graphs).

Lemma

Graph Isomorphism is polynomial time reducible to the problem of computing the automorphism group of a graph.

Proof.

Suppose G_1 and G_2 are two connected graphs. Form their disjoint union $G = G_1 \cup G_2$. Then G_1 and G_2 are isomorphic iff the automorphism group of G contains a map that swaps G_1 and G_2.

In fact, there must be a generator with this property. □