Fixed Parameter Algorithms

Given a problem Π, and also some parameter k, the problem is FPT with parameter K if it can be solved in time $f(K).poly(I)$ where I is the instance size.

So if n is graph size, k is the vertex cover size then

- n^k is not FPT
- $f(k).n^{100}$ is FPT
- $poly(n,k)$ is definitely FPT.

Really a function of the parameter: eg graph coloring with parameter k = color set size or clique, size of clique is not believed to be solvable in time n^k even. ~~But if~~ is parameter, then it is believed that $n^{\Omega(k)}$ is close to best, but maybe with some other parameter we can do better. E.g. with parameter = max degree Δ, there is a $(2^{\Delta}.\Delta^2 n)$ time algo for clique.

OK. Several techniques. Also a lot of crossover with exact (fast) algorithms for NP-hard problems. Will see some of them in these 2 lectures

---

1. Kernelization : apply simple "reduction" rules to get an instance in time $poly(n,k)$ that is of size $g(k)$. on which we can use a brute force algo to get $poly(n,k)+f(k)$ algo.

if $g(k) = poly(k)$ this is "poly sized kernel".

Notation : reduction rule gives $(I,K) \Rightarrow (I',K')$ and is "safe" or "sound" if (I,K) is a YES instance $\Leftrightarrow$ (I',K') is a YES instance. denoted by $(G,k) \Leftrightarrow (G',k')$

Let's take a simple example. Vertex Cover. (K = OPT VC size) Here are two simple reduction rules.

VC1 if v is isolated vertex, drop it; not in any VC.

$(G,K) \Leftrightarrow (G - \text{isolated vertex } v, K)$.

VC2 if v has degree $> k$ then v is in every VC.

$(G,k) \Leftrightarrow (G-v, k-1)$.

<u>This means</u>: When no more reduction rules apply, graph is of min-degree 1 and max degree $\Delta = K$.

<u>Fact</u>: A vertex cover of such a graph (of size K) can cover at most K^2 edges.

So at end: if G^{final} has more than $(K^{final})^2$ edges, $\tilde{G}$ = matching with $K+1$ edges. (disjoint)
else $\tilde{G} = G^{final}$.

Follows that $(G,K) \Leftrightarrow (\tilde{G}, K^{final})$.
and $\tilde{G}$ has $\leq (K^{final})^2$ edges ~~and~~
$\leq K^2$ and hence $\leq 2K^2$ vertices.

<u>Thm</u>: VC has a kernel of size $\leq 2k^2$ vertices (k^2 edges).

Can we do better? YES :- (and no)

[No? Dell & van Melkebeek STOC 10 showed that unless $coNP \subseteq NP/poly$ VC cannot have $O(k^{2-\epsilon})$ edge kernels]

<u>Thm</u>: VC has kernel of size $\leq 2k$ vertices.

<u>Pf</u>: Write LP for VC.
$$\min \sum x_v \quad \text{st } x_u + x_v \geq 1 \ \forall u \sim v, \quad x_u, x_v \geq 0.$$

<u>HW6</u>

<u>Thm</u>: [Nemhauser-Trotter] Any basic feasible solution to this LP has
(a) $x_v \in \{0, 1/2, 1\}$ for all vertices
And if x^* is optimal fractional soln with $V_0 = \{v : x^*_v = 0\}$
$V_{1/2} = \{v : \ — 1/2\}$
$V_1 = \{v : x^*_v = 1\}$.
then ~~OPT~~ $\exists$ optimal solution OPT st
$V_0 \cap OPT = \emptyset$, $V_1 \subseteq OPT$ and OPT contains none of the rest.

<u>Note</u>: $OPT_{LP} \leq |OPT|$. $\Rightarrow |V_{1/2}| \leq 2|OPT| = 2K$.

So reduction rule: drop V_0, pick V_1, $(G,K) \Leftrightarrow (G \setminus (V_0 \cup V_1), K - |V_1|)$.

And size of new instance $\leq 2K$ vertices! ☺

Can solve LP using matchings in this case

Kernelization is a good way to do things, and it is "complete" in some sense.

Thm: problem Π with parameter k is FPT $\Leftrightarrow$ it has a ~~poly-sized~~ kernelization algo.

Pf: one direction is immediate. other side:- sps it has $f(k)\, n^c$ algo. then

either instance size $\leq f(k)$ then output instance.

else size $\geq f(k)$ then run algo on it taking time $f(k)\, n^c \leq n^{c+1}$.

and output a trivial instance depending on answer. ∎

But this is not an interesting thm. Would like poly sized kernels. And that's the best goal to aim for. But let's look at other techniques.

Aside: Sunflower lemma [Erdos-Rado].

gives:- d-uniform hypergraph vertex cover (aka d-hitting set) has kernel with at most $d!\,k^d$ sets.

II Bounding the Branching Factor in Exhaustive Search.

• Observation: in each VC solution, either a node is in OPT or all its nbrs are in OPT.

$$\text{So } Alg(G,k) = \min\left(Alg(G-v, k-1),\ Alg(G-N(v)-v,\ k-|N(v)|)\right)$$

And when max degree of nodes $= 1$, problem is trivial (graph is a matching).

Algo: branch by picking highest degree vertex (degree ≥ 2)

$$\Rightarrow\quad T(k) \leq T(k-1) + \underbrace{T(k-2)}_{N(v)\geq 2} + O(m).$$

Solve to get $\boxed{T(k) \leq O(m).\phi^k}$ $\leftarrow$ 1.618.... golden ratio.

• Easy improvement: what if graph has max degree $= 2$. Paths and cycles. Also easy.

So can stop when min degree $= 2$.

$$T(k) \leq T(k-1) + T(k-3) + O(m)$$

solves to $T(k) \leq O(m).(1.46)^k$.

Can combine with kernelization to get (time for kernelization) $+ O(k^2)(1.46)^k$. ∎

Branch factor was studied for early SAT problems exactly (DPLL procedures)

(Davis Putnam (Logemann Loveland))

Exact $(1.839)^n$ algo for 3SAT:-

If ∄ a 3 variable clause then use 2SAT algo to check satisfiability.

Else $(l_1 \vee l_2 \vee l_3)$:-

either l_1 is true or
l_1 F l_2 T or
l_1 F l_2 F l_3 F

$\Rightarrow T(n) \le T(n-1) + T(n-2) + T(n-3) + O(m+n)$. (# variables)

$T(n) \le \alpha^n \cdot O(m+n)$

α is the real root of $\alpha^3 = \alpha^2 + \alpha + 1$
$\alpha \simeq 1.839\ldots$

Many improvements: Monien Speckenmeyer '85 ,, Schöning '99
Paturi Pudlak Zane (+ Saks), Williams etc.

Another (better) one: Suppose a^* is a sat'ing assignment.

Then suppose #1's in a^* is $\le n/2$ (else start with the all 1's solution)

- start with all zero solution:
- while current solution a is not sat'ing, ~~try~~ pick unsat'd clause
  ~~try each~~ recurse with $a \oplus e_i$ for each i in that clause.

- recursion depth $\le n/2$.

total time $\le 3^{n/2} \cdot O(m+n) = 1.732^n \cdot O(m+n)$ better than above, but not best possible (by any means).

[Schöning 99]: Pick the first clause not sat'd by a, and flip a random variable and repeat. [on opposite side]
for depth $T = O(n)$.

Claim: $\Pr[\text{success}] \ge \frac{\Omega(1)}{\sqrt{n}} (3/4)^n$. $\Rightarrow$ repeat $O(4/3)^n \cdot \text{poly}(n)$ times. $\simeq (1.33)^n$.

[PPZ] deals with unit clauses: random permutation on variables, random "default" assignment.

If current variable is in some unit clause, sat that clause (if no way to sat all unit clauses, abort).
Else set current var to the default value,
and simplify and move to next variable in random order. $(1.58)^n$ etc.

Color Coding: [Alon Yuster Zwick] powerful yet simple technique to obtain algos via randomization. Eg. easy algorithm to find paths of length k in time $O(k!)$ and $O((2e)^k)$. See HWs.

Inclusion Exclusion:

(can be used to find if $\exists$ a Hamilton cycle.)

- Want to find TSP on a given metric. Trivial sol^n: $n!\ \text{poly}(n)$.
  $(n-1)!\ \text{poly}(n)$.

just basic DP.

starts at s.

$$\underbrace{TSP(\text{ends at } t, \text{ visits vertices } U)}_{\substack{\text{a path from } s \text{ to } t \\ \text{having visited vertices in } U \\ \text{exactly once.}}} = \min_x \left\{ TSP(\text{ends at } x, \text{ visits vertices } U-t) + d(x,t) \right\}$$

takes time (and space) $O(2^n \cdot \text{poly}(n))$

Actually space $O(2^n \cdot n)$
time $2^n \cdot n \times n$ per table lookup.

Suppose now we want to count # of solutions (Hamilton cycles).

Let A_u be the ~~length~~ set of all n-length "closed" walks that ~~start~~ start at s and pass through vertex $u \in V \setminus s$

Note that $|HAM| = \left|\bigcap_{u \in V\setminus s} A_u\right|. = \sum_{X \subseteq V\setminus s} (-1)^{|X|} \left|\bigcap_{i \in X} \bar{A}_i\right|$

eg. $|A \cap B|$
$= |U| - |A| - |B| + |\bar{A} \cap \bar{B}|$
etc.

see over

$\Rightarrow$ easier problem: $\left|\bigcap_{u \in X} \bar{A}_u\right| = $ # of closed walks (of length n) within $(V \setminus X)$. (starting from s.)

can be computed using "matrix multiplication" type ideas.
compute $(A|_{V\setminus X})^n$.

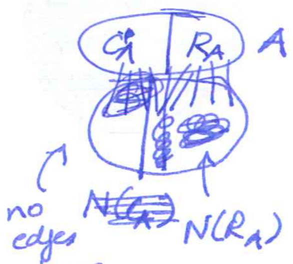
$\Rightarrow$ compute all simpler problems in time $O(n^3 \cdot 2^n)$

but now space polynomial!!

Iterative Compression: [Reed Smith Vetta]

Start of Idea

Vertex cover: sps we have A = approx VC of size $\leq 2K$.



$\Leftarrow A$ is a VC

① guess how $C^* \cap A$ looks like (call this G_A).

R_A = rest of A = $A \setminus G_A$.

- if $G[R_A]$ has an edge, fail for this G_A.
- else $N(R_A)$ must be all in the C^*. (Else some edge not covered)

$N(R_A) = \boxed{\text{Neighbors of } R_A \text{ not in } A.}$

$\Rightarrow$ if $|N(R_A)| + |G_A| > k$ fail for this G_A.

$\Rightarrow$ if ~~$\nexists$~~ $\exists\, G_A \subseteq A$ st $|G_A| + |N(R_A)| \leq k$ this is a VC of size $\leq k$

else fail.

Time: $O^*(2^A) \leq O^*(4^k)$.

Much worse than previous ideas. Can we improve? YES. (~~but get rid of~~ iterative compression.

Iterative Compression

Let $G_i = G[v_1 \ldots v_i]$.

Sps we have a VC of G_i, called A_i, of size K.

then get a VC A_{i+1} of G_{i+1} of size $k+1$.

Now use the algo from previous part to get VC C_{i+1} of size k

in time $O^*(2^{|A_{i+1}|}) = O^*(2^{k+1})$.

$\Rightarrow$ whole thing takes $O^*(2^k)$ ~~poly~~ time! ☺.

loss is a factor n in the runtime.

BTW: very general, but why did it work? We had a sol^n A, we said that we wanted to extend G_A ~~to~~ another soln of size k, but it had to be disjoint from $R = A \setminus G_A$. This gave a lot of structure, make it trivial in the VC case. But this disjointness requirement makes it easy in other cases too.

Longest path in $O^*(2^k)$ time

Want a polynomial $P(\bar{x}) \not\equiv 0$ iff G has a k-path.

for every k-walk $v_1 e_1 v_2 e_2 \dots e_{k-1} v_k$ in the graph we have a

monomial $\prod_{i=1}^{k} y_{v_i} \prod_{i=1}^{k-1} x_{e_i}$ $\left\{ \begin{array}{l} x_e \text{ for edges} \\ y_v \text{ for vertices.} \end{array} \right.$

note: for a path, the monomial is multilinear (each variable has degree ≤ 1)

so instead of checking for existence of some monomial, want to check existence of multilinear monomial.

Idea: Label vertices with a unique label. Now define polynomial over labeled walks.

W = walk
ℓ = labeling $[k] \to [k]$ bijection.

$\forall e=(u,v) \in E, \quad x_{uv}$
$\forall v \in V, \forall a \in [k] \; y_{v,a}$

$$P(\bar{x}, \bar{y}) = \sum_{\substack{\text{walks } W \\ =(v_1, \dots v_k)}} \sum_{\substack{\text{labelings} \\ \ell [k] \to [k] \\ \text{bijection}}} \prod_{i=1}^{k-1} x_{v_i, v_{i+1}} \prod_{i=1}^{k} y_{v_i, \ell(i)}. \qquad // \text{def.}$$

$\bar{x}$: $|E|$ vars; $\bar{y}$: $|V| \times k$ vars

$$= \sum_W{}' \sum_\ell{}' \; MONO_{W,\ell}(\bar{x}, \bar{y})$$

View as a multivariate polynomial over $GF(2^{\log 4k})$.

$\boxed{\text{Crucial Facts}}$

* $GF(2^t)$ has characteristic 2 (i.e. $a+a=0 \; \forall a \in GF(2^t)$)
* has a correspondence with t-bit vectors, but the operations are based on irreducible polys. See an algebra book

---

$\boxed{\text{Main Lemma}}$:

note: not sum over walks but over paths

$$P(\bar{x}, \bar{y}) = \sum_{\substack{\text{paths} \\ P}}{}' \sum_\ell{}' \; MONO_{P,\ell}(\bar{x}, \bar{y}).$$

We will see its proof soon, but for now let's use it.

note that for two different paths P, P' $MONO_{P,\ell}(\bar{x},\bar{y})$ cannot cancel $MONO_{P',\ell'}(\bar{x},\bar{y})$ for any ℓ, ℓ'. (since the x parts are just different).

$\Rightarrow$ Corollary: P is non zero polynomial iff $\exists$ a k-path in G.

How to compute P at $(\bar{x},\bar{y})$ efficiently?

$\boxed{\text{Evaluation lemma}}$: Given assignments to $\bar{x},\bar{y}$, $P(\ )$ can be calculated in time $O(2^k \cdot km)$ field operations.

Thm: $\exists$ an algorithm for LONGEST path in time $O^*(2^k)$.

Pf: want to check if $P(\ )$ is nonzero polynomial.
(by Corollary $\Leftrightarrow$ G has a k path).
Use schwartz-zippel, evaluation lemma says takes time $O^*(2^k)$ to evaluate it at random points. And if $P() \not\equiv 0$ then whp this will give non zero value. ∎

How to prove the Lemmas:

① Proof of Main Lemma: let W be a walk (non-path) and ℓ bijection.
Say v is repeated, the first one. Flip the labels of those two locations, to get ℓ'. note $MONO_{W,\ell} = MONO_{W,\ell'}$. hence they'll cancel each other out (b/c $GF(2^t)$ has characteristic 2). ∎

② Proof of Evaluation Lemma:

N.b. Wtd inclusion Exclusion: $w(\bigcap_{i\in n} A_i) = \sum_{X\subseteq[n]}' (-1)^{|X|} w(\bigcap_{j\in X} \bar{A}_j)$ $w(\bigcap_{j\in\emptyset}\bar{A}_j) = w(U)$.

for any weight f^n.

Claim 1: $\forall$ walks $W, \bar{x}, \bar{y}$

$$\sum_{\substack{\ell \text{ bijection} \\ [k]\to[k]}}' MONO_{W,\ell}(\bar{x},\bar{y}) = \sum_{X\subseteq[k]}' \sum_{\substack{\ell:\text{maps} \\ \text{from } [k]\to X}}' MONO_{W,\ell}(\bar{x},\bar{y})$$

Pf: Let U be set of all maps from $[k]$ to $[k]$.

A_i = set of maps that map at least one thing to i

Fix $W, \bar{x}, \bar{y}$

$w(\ell) = MONO_{W,\ell}(\bar{x},\bar{y})$.

$$\text{we want LHS} = w(\bigcap_{i\in k} A_i) = \sum_{X\subseteq[k]}' (-1)^{(X)} w(\bigcap_{j\in X} \bar{A}_j)$$

$$= \sum_{X\subseteq[k]}' (-1)^X \sum_{\substack{\ell:\text{maps from} \\ [k] \text{ to } [k]\setminus X}}' MONO_{W,\ell}(\bar{x},\bar{y})$$

$\boxed{\text{Char } 2}$

$$= \sum_{X\subseteq[k]}' \sum_{\substack{\ell:\text{maps from} \\ [k] \text{ to } X}}' MONO_{W,\ell}(\bar{x},\bar{y}).$$

∎

$$\Rightarrow P(\bar{x},\bar{y}) = \sum_{\text{walk } W}' \sum_{X\subseteq[k]}' \sum_{\substack{\ell:\text{maps from} \\ [k]\to[X]}}' MONO_{W,\ell}(\bar{x},\bar{y}).$$

$$= \sum_{X\subseteq[k]}' P_X(\bar{x},\bar{y}). \quad = 2^k \text{ polynomials of the form } \sum_{\text{walks}}' \sum_{\substack{\text{maps from} \\ k\to X}}' MONO_{W,\ell}(\bar{x},\bar{y}).$$

Claim 2: $P_X(\bar{x},\bar{y})$ can be computed in time $\mathrm{poly}(n,k)$.

Pf: Just a simple DP. on the length of the walk.

length 1: $M(v,1)$ $\sum_{\substack{\text{labelings} \\ \text{from } [k]\to X}}' y_{v_0,\ell(k)}$ and then

$$M(v,i) = \sum_{\substack{\text{u}\to v, \\ \text{label for } k-i+1}}' y_{v,\ell(k-i+1)} \cdot x_{vu} M(u,i-1) \text{ etc.}$$

$O(k^2n^2)$ time.